БИБЛИОТЕЧКА ПРОГРАММИСТА

Ю. М. БЕЗБОРОДОВ

От фортрана к PL-1

БИБЛИОТЕЧКА
ПРОГРАММИСТА

Ю. М. БЕЗБОРОДОВ

ОТ ФОРТРАНА—К PL/1

ОСНОВЫ ЯЗЫКА PL/1

МОСКВА «НАУКА»
ГЛАВНАЯ РЕДАКЦИЯ
ФИЗИКО-МАТЕМАТИЧЕСКОЙ ЛИТЕРАТУРЫ
1984

От фортрана — к PL/1. Основы языка PL/1. Безбородов Ю. М.— М.: Наука. Главная редакция физико-математической литературы, 1984.— 208 с.

Книга является курсом универсального языка программирования PL/1, излагаемого на базе языка фортран. Читателям, знакомым с фортраном, книга поможет быстро овладеть основными понятиями PL/1 и приступить к программированию на этом языке, используя имеющиеся у них навыки работы на фортране.

Юрий Михайлович Безбородов

ОТ ФОРТРАНА — К PL/1. Основы языка PL/1

(Серия: «Библиотечка программиста»)

Редактор *Л. Г. Силкова*

Техн. редактор *Л. В. Лихачева.* Корректор *С. Н. Макарова*

ИБ № 12428

Сдано в набор 26.09.83. Подписано к печати 07.02.84. Т-06446. Формат 84×108¹/₃₂. Бумага тип. № 3. Литературная гарнитура. Высокая печать. Условн. печ. л. 10,92. Услови. кр.-отт. 11,13. Уч.-изд. л. 12,22. Тираж 80 000 экз. Заказ № 2257. Цена 75 коп.

Издательство «Наука». Главная редакция физико-математической литературы 117071, Москва, В-71, Ленинский проспект, 15

Ордена Октябрьской Революции и ордена Трудового Красного Знамени Первая Образцовая типография имени А. А. Жданова Союзполиграфпрома при Государственном комитете СССР по делам издательств, полиграфии и книжной торговли. Москва, М-54, Валовая, 28.

Отпечатано в Подольском филиале ПО «Периодика» Союзполиграфпрома при Государственном комитете СССР по делам издательств, полиграфии и книжной торговли. г. Подольск, ул. Кирова, д. 25.

Б $\frac{1702070000-048}{053\ (02)-84}$ 35-84

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ

В связи с внедрением в народное хозяйство вычислительных машин Единой системы (ЕС ЭВМ) широкое распространение в последние годы в Советском Союзе получает универсальный язык программирования PL/1. Язык PL/1 [5—8] предназначен как для решения задач численного анализа, так и коммерческих задач, и задач обработки символьных и двоичных данных; в языке предусмотрены средства для работы со списками данных, с наборами данных, имеющими самую различную организацию. Наличие в PL/1 блоков и групп операторов, а также структур (записей) позволяет при разработке программ успешно применять современные методы нисходящего и структурного программирования [1—4].

В настоящее время основным языком для решения массовых научно-инженерных задач в Советском Союзе является, по-видимому, фортран в одной из существующих версий (фортран II — фортран IV — фортран ЕС). В частности, на базе именно языка фортрана в подавляющем большинстве технических вузов студенты изучают основы программирования. Но в фортране нет удобных средств для решения перечисленных выше задач (за исключением задач численного анализа), что препятствует использованию этого языка в ряде областей науки и техники. Кроме того, набор и строение операторов языка фортран, появившегося впервые еще в 1956 году, уже не отвечают современным требованиям и затрудняют применение методов структурного и нисходящего программирования.

В настоящей книге преследуется цель облегчить изучение основ языка PL/1 для читателей, знакомых с языком фортран в какой-либо из версий [9—14], и, в особенности, для тех из них, кто начинает работать в ОС ЕС ЭВМ. Поскольку многие основные понятия PL/1 совпадают с понятиями фортрана или являются их развитием, оказывается возможным излагать PL/1, непосредственно привлекая при этом имеющиеся у читателя знания фортрана. Изложение сложных понятий PL/1 дается в два этапа: сначала указываются их общие с фортраном свойства, а затем уже описываются новые или допол-

нительные по сравнению с фортраном возможности. Все это должно позволить читателям значительно ускорить и облегчить освоени многих понятий PL/1, имеющих аналоги в фортране. (Читатели, ознакомившиеся с первыми четырьмя главами и п. 5.1, уже смогут составлять реальные одномодульные программы; изучение всей главы 5 позволит им составлять многомодульные программы.) Для ряда отсутствующих в фортране IV понятий (или возможностей) PL/1 дается их краткое сравнение с понятиями нового стандарта языка фортран — фортраном 77 [15].

Книга состоит из введения, девяти глав и приложения. Во введении дается краткая характеристика языка PL/1 и общее сравнение его с фортраном IV. В главах 1—6 рассматриваются основные понятия PL/1 (основные элементы, выражения, операторы, описания, процедуры, ввод-вывод). В главах 7 и 8 излагаются некоторые дополнительные возможности и понятия PL/1, не имеющие аналогов в фортране. В главе 9 собран ряд сведений, имеющих справочный характер. Приложение (написанное Т. Г. Лаврентьевой) содержит описание правил взаимосвязи программных модулей, составленных на PL/1 и фортране ЕС; правила в ряде случаев позволят читателям при разработке новых программ на PL/1 использовать старые модули, написанные на фортране. Первые 8 глав снабжены упражнениями, ответы к которым приведены в конце книги. Заметим, что ссылки вперед, которые встречаются в книге, при первом чтении могут быть опущены читателем без особого ущерба для понимания излагаемого понятия. При изучении главы, посвященной вводу-выводу, от читателя потребуются минимальные знания, касающиеся параметров директивы DD языка управления заданиями [18, 19].

В настоящей книге язык PL/1 описывается на уровне входного языка транслятора версии F для ОС ЕС ЭВМ. Оптимизирующий [5, приложение 13] и отладочный трансляторы обеспечивают цел ы ряд дополнительных возможностей по сравнению с приведенными в этой книге.

Автор считает своим приятным долгом выразить благодарность Н. П. Трифонову и А. С. Маркову за проявленный интерес к работе и полезную критику. Автор признателен также А. Х. Алдакову Н. Т. Быстровой и О. Ф. Репкиной, оказывавшим постоянную помощь в работе над рукописью.

Ю. М. Безбородо

«...большинство людей навсегда привязывается к языку, который они изучали первым... Первый язык — удобная форма для конкретного воплощения абстрактной мысли».

Н. Вирт. «Дисциплина программирования».

ВВЕДЕНИЕ

Язык PL/1 (Programming Language — язык программирования) является универсальным алгоритмическим языком. Название «язык программирования» просто подчеркивает, что прямым назначением языка (как, впрочем, и фортрана) является именно запись программ для их выполнения на машине, а не алгоритмов с целью их публикации (что в определенной мере свойственно, например, для алгола-60).

Общее сравнение. В отличие от фортрана язык PL/1 предназначен для решения задач не только численного анализа, но и для научно-инженерных задач широкого профиля, коммерческих задач, задач АСУ, а также задач системного программирования. В PL/1 имеются специальные средства для работы с символьными и двоичными данными, с данными, организованными в списки, в таблицы («структуры»), в разнообразные по формам обработки файлы (наборы данных).

Сравнивая основные средства рассматриваемых языков, можно, пожалуй, сказать, что в PL/1 отсутствуют лишь следующие две существенные возможности фортрана: (1) представление списка форматов (в операторах ввода-вывода) в виде массива или строки символов и формирование такого списка другими операторами ввода-вывода; (2) возможность изменения правил умолчания *) (оператор IMPLICIT). Заметим, что обе эти возможности опущены в PL/1, очевидно, в силу того, что они противоречат требованиям структурного программирования: они нарушают статичность текста программы и тем самым затрудняют ее проверку.

Все остальные основные понятия и возможности фортрана в том или ином виде присутствуют в PL/1. В частности, можно сказать,

---

*) Такая возможность имеется в оптимизирующем трансляторе PL/1 [5, приложение 13].

что строение арифметических и логических выражений, а также возможности ряда операторов, например, вызова (CALL), перехода (GO TO), останова (STOP) в обоих языках одни и те же. Аналоги других операторов фортрана обладают в PL/1 более широкими возможностями. Так, условный (IF) оператор PL/1 может включать в себя не только один оператор, но и группу операторов; в нем же указываются и операторы, выполняемые в случае, когда условие ложно. Заголовок оператора цикла в PL/1 может содержать дополнительное условие продолжения выполнения цикла; параметр цикла может принимать и отрицательные и вещественные значения, а его начальное и конечное значения, а также шаг приращения могут быть заданы произвольными выражениями. Длины полей в форматах операторов ввода-вывода также могут задаваться выражениями, а не только константами, как в фортране.

Из дополнительных по отношению к фортрану возможностей PL/1 отметим наличие операторов и операций, выполняющих действия сразу над всеми соответствующими элементами массивов-операндов (в частности, в выражениях и при присваивании). Такая же возможность имеется и для новых величин — *структур*, объединяющих данные различных типов и организаций. Помимо последовательных наборов данных (бесформатный ввод-вывод в фортране) в PL/1 реализован и ряд других видов наборов данных, позволяющих при решении конкретных задач выбрать необходимый способ обработки внешних данных, обеспечивающий наибольшие удобства и эффективность.

Новыми по сравнению с фортраном являются также понятия блока и ситуаций. Понятие блока позволяет, в частности, вводить независимую систему обозначений для используемых величин не только в целых подпрограммах, как в фортране, но и в отдельных частях подпрограмм. Ситуации и связанные с ними операторы позволяют программистам задавать свою процедуру обработки ошибок, которые могут проявиться в ходе выполнения программы, что помогает успешному проведению отладки.

В PL/1 имеется и ряд других возможностей (параллельное выполнение независимых частей программы, макрогенерация текста программы [17] и др.), но они в настоящей книге не затрагиваются.

Примеры. Ниже приведены примеры учебных программ на фортране и PL/1, состоящих из двух модулей и выполняющих одни и те же вычисления. В примерах из упорядоченного по неубыванию вектора V и скаляра R образуется вектор W с той же упорядоченностью; кроме того, определяется место величины R в векторе W (характеризуемое индексом элемента). Примеры позволяют более конкретно продемонстрировать и сходство и некоторые отличия программ на PL/1 и фортране.

Пример на фортране.

```
C     ГОЛОВНОЙ МОДУЛЬ: ВВОД—
C     ОБРАЩЕНИЕ К INSERT—ВЫВОД
      REAL V(40), W(41), R
      INTEGER K
00001 READ(5, 10) V, R
   10   FORMAT(5(8E10.3/), E10.3)
      CALL INSERT(V, 40, W, 41, R, K)
      WRITE(6, 20) K, W
   20   FORMAT(1X, 'МАССИВ␣W␣ДЛЯ K=', I5, 6(/8E12.3))
      STOP
      END
C     ПОДПРОГРАММА ОБРАЗОВАНИЯ
C     УПОРЯДОЧЕННОГО МАССИВА Y
C     ИЗ УПОРЯДОЧЕННОГО ПО НЕУБЫВАНИЮ
C     МАССИВА X И ВЕЛИЧИНЫ A
      SUBROUTINE INSERT(X, N, Y, M, A, K)
      INTEGER N, M, I, K
      REAL X(N), Y(M), A
C     ПЕРЕСЫЛКА ДЛЯ X(I) < = A
00001 DO 10 I=1, N, 1
        K=I
        IF (X(I) .GT. A) GO TO 20
        Y(I)=X(I)
   10 CONTINUE
C     ПЕРЕСЫЛКА A
        K=K+1
   20   Y(K)=A
C     ПЕРЕСЫЛКА ДЛЯ X(I) > A
        IF (K .GT. N) GO TO 40
        DO 30 I=K, N
   30     Y(I+1)=X(I)
   40   RETURN
      END
```

Пример на PL/1:

```
/*    ГОЛОВНОЙ МОДУЛЬ: ВВОД—ОБРАЩЕНИЕ
                                    К INSERT—ВЫВОД */
PRIMER: PROCEDURE OPTIONS(MAIN);
          DECLARE(V(40), W(41), R) FLOAT, K FIXED;
          DECLARE INSERT ENTRY( FIXED, , );
    ○○○: GET EDIT(V, R) (5(8 E(10, 3), SKIP), E(10,3));
          CALL INSERT(V, 40, W, R, K);
          PUT EDIT('МАССИВ␣W␣ДЛЯ␣K=', K, W) (R(L));
       L:    FORMAT(A, F(5), 6(SKIP, 8E(12,3)));
```

```
END PRIMER;
* PROCESS;
/*  ПОДПРОГРАММА ОБРАЗОВАНИЯ
                              УПОРЯДОЧЕННОГО МАССИВА Y */
/** ИЗ УПОРЯДОЧЕННОГО ПО НЕУБЫВАНИЮ
                              МАССИВА X И ВЕЛИЧИНЫ A **/
INSERT: PROCEDURE(X, N, Y, A, K);
        DECLARE(X(*), Y(*), A) FLOAT, (N, I, K) FIXED;
  ##:   DO K=1 TO N BY 1 WHILE (X(K) <=A);
          Y(K)=X(K); /* ПЕРЕСЫЛКА ДЛЯ X(K) <=A */
        END; /** ЗНАЧЕНИЕ ПАРАМЕТРА
                              ЦИКЛА СОХРАНЯЕТСЯ! **/
        Y(K)=A; /* ПЕРЕСЫЛКА A */
        DO I=K TO N; Y(I+1)=X(I); END;
                              /* ПЕРЕСЫЛКА ДЛЯ X(I) > A */
END INSERT;
```

Как видим, по сравнению с фортраном, одни операторы PL/1 очень похожи, другие видоизменены (например, укрупнены); ряд операторов обладают дополнительными возможностями. В одном операторе описания PL/1 могут описываться величины различных типов. Список форматов можно задавать и внутри самого оператора ввода-вывода. Комментарии могут записываться и в одной строчке с операторами.

Дополнительное по сравнению с фортраном описание имени подпрограммы бывает необходимо в том случае, когда среди аргументов в обращениях к подпрограмме присутствуют константы. Метками ⬡⬡⬡ и ## выше помечены первые выполняемые операторы модулей.

Обращает на себя внимание оператор цикла в PL/1. В его заголовке может быть указано дополнительное условие продолжения выполнения цикла. Кроме того, перед началом выполнения тела цикла в PL/1 осуществляется сравнение начального значения параметра цикла с конечным значением и в необходимых случаях — обход выполнения тела цикла (для второго оператора цикла, например, при K > N).

Заметим, что параметр (фиктивный аргумент) M, фигурирующий в подпрограмме на фортране в данном случае (с одномерными массивами) не является обязательным, если использовать описание вида

```
REAL X(1), Y(1), A
```

В PL/1 использование звездочек при описании массивов-параметров позволяет всегда опускать параметры, задающие границы массивов.

Конечно, подпрограмма INSERT на PL/1 при желании могла бы быть написана и в виде, максимально конирующем фортранный стиль. Например:

```
/* ПЕРЕСЫЛКА ДЛЯ X(I) < = A */
  #00001: DO I=1 TO N;
              K=I;
            IF(X(I) > A) GO TO #20;
              Y(I)=X(I);
          END;
/* ПЕРЕСЫЛКА A */
          K=K+1;
     #20: Y(K)=A;
  и т. д.
```

Но, очевидно, что только использование новых, более мощных операторов PL/1 позволит программисту повысить эффективность своей работы при программировании на этом языке.

После рассмотрения приведенных примеров дополнительно к сказанному ранее можно дать следующую сравнительную характеристику основных операторов двух языков. В фортране операторы более простые и легко осваиваются начинающими программистами; эта простота приводит обычно и к получению более эффективных, чем в PL/1, объектных программ после трансляции. Но процесс алгоритмизации и программирования задачи на фортране происходит, обычно, труднее, чем на PL/1, что приводит к большему количеству ошибок и к сравнительному удлинению сроков отладки программ (особенно для сложных задач).

Основные обозначения. Описание синтаксиса языка PL/1 в книге будет даваться с применением синтаксических форм. В формах используется, в основном, система обозначений (метаязык), принятая в документации ОС ЕС ЭВМ. Обозначения разделяются на метапеременные (обозначения понятий), метаконстанты (литералы), метасимволы (конструктивные обозначения).

Метапеременные, служащие для обозначения понятий языка, удут представляться обычными русскими словами (может быть, сокращенными), состоящими из строчных букв. Когда понятие представляется несколькими словами, они будут соединяться дефисами; в этом случае в качестве поясняющих слов могут использоваться, например, и ключевые слова. Примеры:

метка
константа-арифметическая
оператор-END
оператор-присваивания

Метаконстанты представляются одним символом или сочетанием символов, допустимых в описываемом языке, например:

E
5
CALL

Метасимволы служат для указания порядка следования элементов (метапеременных и метаконстант) в некоторой конструкции языка. В дальнейшем используются следующие метасимволы (они не могут являться символами описываемого языка):

$\sim$ — «соответствует», «это». В синтаксических формах слева от этого символа помещается определяемое понятие языка, справа — конструкция из метаобозначений, определяющая это понятие или эквивалентная ему. Например (здесь и ниже примеры пока приводятся для фортрана):

оператор-END $\sim$ END

| — «или». Используется внутри метаскобок { }, [ ] или ( ) (см. ниже).

{ } — выбор (альтернатива). Например:

$$\textit{константа-логическая} \sim \left\{\begin{matrix}\text{.TRUE.}\\ \text{.FALSE.}\end{matrix}\right\} \sim \{\text{.TRUE.} \mid \text{.FALSE.}\}$$

$$\textit{знак-операции-арифметической} \sim \{+ \mid - \mid * \mid / \mid **\}$$

В сочетании с метасимволами повторения (см. ниже) данные скобки иногда употребляются для выделения какой-либо группы элементов конструкции в качестве одного ее элемента, т. е.

$$\{xyz\} \sim xyz$$

Здесь x, y и z — некоторые произвольные конструкции определяемого языка. Справедлива, например, форма:

$$x\{y \mid z\} \sim \{xy \mid xz\}$$

[ ] — необязательный выбор или необязательный элемент в конструкции — *опция* *). Например:

оператор-CALL $\sim$
$\sim$ CALL *имя-подпрограммы* [(*список-аргументов*)] $\sim$
$\sim$ {CALL *имя-п/п* (*список-аргументов*) | CALL *имя-п/п*}

Справедливы, например, формы:

$[x]\,y \sim \{xy \mid y\}$
$x\,[y \mid z] \sim \{xy \mid xz \mid x\}$

*) От английского option — выбор.

⟨ ⟩ — уточнения и комментарии в синтаксической форме; они не являются неотъемлемой частью какой-либо конструкции, например (см. также примеры ниже):

$x\langle y\rangle z \sim xz$

... — повторение предшествующего элемента. Например:

целое-число ~ [±] *цифра*...
имя ~ *буква* [*буква* | *цифра*]... ⟨до 6 символов⟩

Повторение является более старшей операцией, чем выбор ({ } или [ ] или ( )), то есть:

$$\left\{\begin{matrix}y\\z\end{matrix}\right\}\ldots \sim \left\{\left\{\begin{matrix}y\\z\end{matrix}\right\}\middle|\left\{\begin{matrix}y\\z\end{matrix}\right\}\left\{\begin{matrix}y\\z\end{matrix}\right\}\middle|\left\{\begin{matrix}y\\z\end{matrix}\right\}\left\{\begin{matrix}y\\z\end{matrix}\right\}\left\{\begin{matrix}y\\z\end{matrix}\right\}\langle\text{и т. п.}\rangle\right\}$$

.. — список, то есть перемежающееся повторение двух предшествующих элементов в конструкции:

yz.. ~ *y* [*zy*]...
список-аргументов ~ *аргумент*,..
оператор-EXTERNAL ~ EXTERNAL *имя-подпрограммы*,..

( ) — полуобязательный выбор: в некоторой законченной или повторяющейся конструкции хотя бы один из ее элементов, заключенных в такие скобки, является обязательным, например:

$)\,y\,(z) \sim \{xyz \mid xy \mid yz\}$

равните:

$x]\,y\,[z] \sim \{xyz \mid xy \mid yz \mid y\}$

ример для фортрана:

число-вещественное-без-порядка ~ [±] (*цифра*...).(*цифра*...)

Кроме того, отметим следующий важный случай:

$\{x \mid (y)\}\ldots \sim [x \mid y]\ldots y\,[x \mid y]\ldots$

т. е. хотя бы один элемент *y* будет присутствовать в приведенной конструкции.

⟶ и **⟵** — указывают на факт преобразования или вычисления некоторой величины.

Типографский пробел в формах будет в необходимых случаях отделять одно понятие от другого (или от литерала), но это не значит, что он должен присутствовать и в реальных конструкциях языка. Размещение пробелов в формах указывается явно значком ␣ или регламентируется правилами, существующими в конкретном языке. Последнее справедливо и для правил перехода с одной строчки текста программы на другую.

ГЛАВА 1

ОСНОВНЫЕ ЭЛЕМЕНТЫ ЯЗЫКА

1.1. Символы

В PL/1, как и в фортране, основные символы языка разделяются на буквы, цифры и специальные знаки. Этот факт с помощью синтаксической формы выражается следующим образом:

символ-языка ~ {*буква* | *цифра* | *спецзнак*}

В качестве букв в PL/1, как и в фортране, используются 26 заглавных букв латинского алфавита, но введены еще и 3 дополнительные «буквы»: ▢ (на некоторых устройствах заменяется на $), @, #. В коммерческих расчетах дополнительные буквы имеют следующий смысл: ▢—знак денежной единицы, @—коммерческое «*at*», #—номер. Напомним, что в фортране ЕС знак ▢ (или $) также отнесен к буквам.

Синтаксическая форма для буквы PL/1 будет иметь следующий вид:

буква ~ {A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | | T | U | V | W | X | Y | Z | ▢ | @ | #}

Цифры в PL/1 обычные, но нуль при написании его на бланке (и при печати) обычно перечеркивается (ø) для отличия от буквы О. Итак:

цифра ~ {0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9}

К спецзнакам PL/1, используемым в основных конструкциях языка, относятся следующие:

спецзнак ~ {+ | − | * | / | = | . | , | (|) | ' | ␣ | < | > | & | ! | ¬ | ; | : | _ | %}.

Первые 10 спецзнаков PL/1 имеют тот же смысл, что и в фортране, остальные знаки будут поясняться по мере их применения. Здесь отметим, что в PL/1 на использование знака пробела (представленного в форме символом ␣) наложены гораздо более жесткие требования, чем в фортране; все они будут даны в ходе дальнейшего

изложения (см., например, 1.8). Напомним, что при написании текста программы на бланке знак пробела обычно не указывается, так как каждая незаполненная позиция бланка считается пробелом.

Помимо перечисленных выше символов в некоторых конструкциях языка, таких как строки и комментарии, можно использовать и любые другие символы, имеющиеся на устройствах ввода-вывода конкретной ЭВМ. Для определенности будем считать, что в набор таких символов помимо символов языка входят, в частности, заглавные русские буквы, не совпадающие по начертанию с латинскими буквами:

символ-ЭВМ ~ {*символ-языка* | Б | Г | Д | Ж | З | И | Й | Л | П | У | Ф | Ц | Ч | Ш | Щ | Ь | Ы | Э | Ю | Я}

1.2. Идентификаторы

Идентификатор, или в обычной фортранной терминологии — *имя*, в PL/1 имеет то же строение, что и в фортране, но максимальная его длина равна не 6, а 31 символу. Отметим сразу, что пробелы в именах (идентификаторах) PL/1 не допускаются в отличие от имен фортрана.

Примеры:

A, B15, EPSYLON, ¢CUP, STEEL#40A

Наряду с буквами и цифрами в идентификаторе PL/1 могут присутствовать и «знаки размыкания» (_ или на некоторых устройствах ¬), с помощью которых можно представлять имена в более наглядной форме. Например, вместо имен

ONEORTWO или TOTALCOST

лучше взять другие:

ONE_OR_TWO или TOTAL_COST

Разумеется, TOTALCOST и TOTAL_COST — это различные имена.

Замечание. «Внешние» имена в PL/1 (см. 4.4 и 5.1) могут содержать не более 7 символов, среди которых знаки размыкания и «буквы» ¢, #, @, как правило, не используются.

Формы.

идентификатор ~ *имя* ~
~ *буква* [*буква* | *цифра* | _] . . . <до 31 символа>

имя-внешнее ~
~ *буква-лат* [*буква-лат* | *цифра*] . . . <до 7 символов>

1.3. Ключевые слова

В PL/1, как и в фортране, некоторые идентификаторы, такие, например, как REAL, COMPLEX, IF, DO, RETURN и т. п., используются в качестве ключевых слов во многих операторах PL/1.

Пробелы внутри ключевых слов PL/1 не допускаются; исключением является GO TO.

В PL/1 отсутствует ограничение, имеющееся в фортране, которое требует, чтобы имена, используемые программистом в программе, не совпадали с ключевыми словами языка: предполагается, что транслятор PL/1 сможет всегда отличить ключевые слова от таких же имен переменных, функций и т. п. Для того чтобы не снижать наглядность текста программы, не следует широко использовать упомянутую возможность языка PL/1. (В фортране 77 ключевые слова, как и в PL/1, не резервируются.)

1.4. Метки

Метки в PL/1 имеют вид идентификаторов, а не целых чисел, как в фортране:

метка ~ идентификатор

Метка PL/1 присоединяется к оператору с помощью двоеточия. Например, помеченному оператору присваивания в фортране:

120 X = Y + Z

в PL/1 может соответствовать оператор присваивания (см. 3.1) вида

M120: Y = Y + Z; или XYZ: X = Y + Z;
или X_ASS: X = Y + Z;

Хотя в PL/1 разрешается располагать метки в любых позициях на бланке, имеет смысл, как и в фортране, всегда помещать их поближе к левому краю строчки, чтобы можно было легко их находить в ходе написания и особенно проверки программы.

Замечание. На бланке PL/1 программа может располагаться в позициях 2—72; первая позиция обычно оставляется пустой, а в позициях 73—80, предназначенных для нумерации строчек программы, можно, вообще говоря, помещать любую информацию.

1.5. Переменные

В PL/1 используются переменные различных видов. Здесь мы коснемся только *скалярных* переменных PL/1, а именно, простой переменной и переменной с индексами, для которых имеются прямые аналоги в фортране.

Простая переменная PL/1 соответствует переменной фортрана и представляется в тексте программы идентификатором.

Примеры простых переменных:

I, PI, R105, KMAX, MATRIX_CROSS

Переменная с индексами, как и в фортране, служит для представления элементов массивов, например:

A(2,3), TAU(K,1,L), VECTOR(I/N+1).

В качестве индексов в PL/1 могут использоваться арифметические выражения любого типа и сложности, в частности и такие, которые запрещены в фортране II.

В отличие от ряда версий фортрана индексы в PL/1 могут принимать и нулевые и отрицательные значения. В ходе выполнения программы индексы приводятся всегда к целым значениям, которые по абсолютной величине не должны превосходить $2^{15}-1 \approx 32\,000$. Приведение к целым значениям осуществляется отбрасыванием дробной части, как и в фортране ЕС. Количество индексов не должно превосходить 31.

Формы.

переменная ~ { *переменная-простая* | *переменная-с-индексами* }

переменная-простая ~ *имя*

переменная-с-индексами ~ *имя-массива* (*список-индексов*)

список-индексов ~ *выражение-арифметическое*, . .

Замечания.

1. Помимо рассмотренных скалярных переменных в PL/1 имеются «массивные» и структурные переменные, а также псевдопеременные (см. соответственно п. п. 8.3, 7.2, 8.4). В частности, массивные переменные позволяют выполнять операций сразу над всеми элементами массива (массивов). Например, если A, B, C — массивы, то можно использовать следующие операторы:

A=0; B=C+1; C=A*B;

2. По типам принимаемых значений переменные в дальнейшем будут разделяться на арифметические (числовые), логические и меточные. Тип переменной указывается при ее описании (см. ниже гл. 4). Для массива, кроме того, как и в фортране, задается описатель массива со списком границ индексов (*атрибут размерности* — в PL/1). Значениями меточных переменных являются метки, которые в PL/1 считаются константами.

1.6. Функции

В PL/1, как и в фортране, имеется большое количество встроенных (стандартных) функций, всегда доступных программисту. Список большинства встроенных функций, имеющихся в PL/1, приведен в конце книги (см. 9.1); некоторые функции поясняются в гл. 8.

Дадим список имен встроенных функций PL/1, эквивалентных по своему назначению основным стандартным функциям фортрана: EXP, LOG, LOG10, SIN, COS, TAN, ATAN, SQRT, ABS, MAX, MIN.

В PL/1 присутствуют также следующие функции, имеющиеся в ряде версий фортрана:
SINH, COSH, TANH, MOD.

Отсутствуют функции, вычисляющие arcsin, arccos, ctg.

Заметим, что в PL/1, в отличие от фортрана, одно и то же имя функции обычно используется и для целых, и для вещественных, и для комплексных значений аргументов. Для упомянутых функций, называемых в PL/1 *математическими* функциями (исключая MIN, MAX, MOD), результат всегда получается с порядком (может быть, и комплексный).

Обращение к функции для ее вычисления с заданными значениями аргументов происходит в ходе выполнения программы при вычислении содержащего функцию выражения. Иногда для того, чтобы различать саму функцию, реализованную, например, в программном обеспечении ЭВМ, и функцию, входящую в выражение, последнюю называют *указателем функции*; указатель функции наряду с именем функции обычно содержит и необходимый список аргументов.

Примеры указателей функций:

EXP(L+2), MAX(X, Y, Z).

В PL/1, как и в фортране, можно работать и с функциями, определенными самим программистом. Правила определения таких функций и обращения к ним будут изложены позднее (см. гл. 5).

Форма.

указатель-функции ~ *имя-функции* [(*список-аргументов*)]

1.7. Константы

Константы, служащие для представления в тексте программы постоянных значений, разделяются в PL/1 на арифметические (числовые), строчные и меточные.

1.7.1. Арифметические константы. Для начала будем считать, что в PL/1 так же как и в фортране, имеются числовые (арифметические) константы трех типов: целые, вещественные, комплексные. Пробелы внутри числовых констант в PL/1, в отличие от фортрана, не допускаются.

Целые константы (числа) в PL/1, как и в фортране, имеют вид последовательности десятичных цифр, перед которой может стоять знак плюс или минус, например:

25 —136 +7777 0 —049 10000

Максимальное количество цифр в целой константе — 15, но иногда требуется, чтобы, как и в фортране ЕС, абсолютное значение не превышало $2^{31}-1 \approx 2 \cdot 10^9$.

Форма.

число-целое ~ [±] *цифра* ...

Величины, которые должны принимать только целочисленные значения, характеризуются атрибутом (описателем) FIXED; в фортране целым величинам приписывается тип INTEGER.

Вещественные числа, которые в PL/1 называются числами *с плавающей точкой*, как и в фортране, состоят из цифровой части (мантиссы) и показательной (порядка). Мантисса, так же, как и в фортране, имеет вид целого, дробного или смешанного числа. Порядок, начинающийся с буквы E, должен, в отличие от фортрана, всегда присутствовать в числе.

Примеры вещественных чисел PL/1:

1.8E—3 —85E4 +.1111E+05 1E0

Максимальное количество цифр в мантиссе — 16, в порядке — 2; значение вещественной константы должно, как и в фортране ЕС, по абсолютной величине заключаться в интервале $(10^{-78}, 10^{75})$.

Величины, которые принимают значения с плавающей точкой, в PL/1 описываются с помощью атрибута FLOAT (плавающий); в фортране аналогичным величинам приписывается тип REAL.

Атрибуты FLOAT и FIXED называются в PL/1 атрибутами *масштаба.*

Форма.

число-плавающее ~ *число-фиксированное* E[±] *цифра* [*цифра*]

Форма для фиксированного числа приведена дальше.

Предупреждение 1. Точность представления вещественной константы PL/1 в машине зависит только от количества цифр в ее мантиссе; буква D, используемая в фортране для указания на представление в машине константы с двойной точностью, в PL/1 не применяется. В соответствии с правилами представления данных в ЕС ЭВМ, для того чтобы вещественная константа имела двойную точность, количество цифр в мантиссе должно быть больше 6. Таким образом, например,

числа фортрана	3.2D—4	—1.2345678D3	7.89E—02
PL/1 имеют вид	3.200000E—4	—1.2345678E3	7.89E—02

Предупреждение 2. Десятичная точка в числе PL/1, в отличие от фортрана, не является признаком представления числа в машине в форме с плавающей точкой (в экспоненциальной,

показательной). Как уже говорилось, в PL/1 таким признаком является только наличие порядка числа. Поэтому для того, чтобы числа вида 3.1416, 1.5, 5., .1 (которые при вычислениях на фортране представляются в машине с плавающей точкой) имели в PL/1 ту же форму представления, к ним необходимо приписать порядок, например:

3.1416E0, 1.5E0, 5E0 или 5.E0, 1E—1 или .1E0

Предупреждение 3. Числа без порядка, содержащие дробную часть (и, следовательно, точку), допустимы в PL/1; они принадлежат к классу чисел с *фиксированной точкой*. Диапазон изменения таких чисел значительно меньше, чем чисел с плавающей точкой и по абсолютной величине равен (10^{-15}, 10^{16}), причем требуется, чтобы количество цифр в числе с фиксированной точкой не превышало 15. Числа с фиксированной точкой могут широко применяться в программе на PL/1, но при операциях с ними требуется более жесткий контроль над величиной получаемых результатов, чем для целочисленных величин (об этом подробнее будет сказано ниже, в 8.1).

Формы.

число-фикс ~ [±] (*число-цел-без-зн*) [.] (*число-цел-без-зн*)
число-цел-без-зн ~ *цифра* . . .

Ломаные скобки указывают на то, что число должно содержать или целую, или дробную часть, или обе вместе. Таким образом, целые числа в PL/1 являются частным случаем чисел с фиксированной точкой. Число вида 5. является целым, так как не содержит дробной части.

В программе PL/1 величины, которые должны принимать значения с фиксированной точкой, при описании получают атрибут FIXED (фиксированный). Такие величины, в отличие от величин с плавающей точкой, представляются в ЭВМ абсолютно точно. Целые величины отличаются от величин с фиксированной точкой только атрибутом (описателем) точности (см. ниже), с помощью которого задается и диапазон изменения описываемой величины.

Комплексные числа в PL/1 представляются в виде, более похожем на обычное математическое изображение, чем в фортране. Сравните, например:

1.55E6—3.80E—4I и (1.55E6, —3.80E—4)

В PL/1 снято ограничение, имеющееся в фортране и требующее, чтобы действительная и мнимая части комплексного числа были представлены в виде вещественных чисел (в смысле фортрана) и имели одинаковую точность. Например, в PL/1 допускаются комплексные числа и следующего вида:

1+25I, 4—1.56I, —7—88.8E—4I и т. д.

В PL/1 допускаются и мнимые числа, например вида 7.1I или —3E4I, являющиеся частным случаем комплексных чисел.

Комплексные числа могут фигурировать в различных конструкциях языка PL/1, причем, в одних случаях допускается ставить пробелы внутри числа рядом со знаком + или —, соединяющим действительную и мнимую части, а в других случаях (например, при вводе) запрещается. Чтобы меньше ошибаться в дальнейшем, имеет смысл не помещать пробелы внутри комплексного числа; ограничения на использование пробелов в комплексном числе даются в дальнейшем по ходу изложения.

Величины PL/1, которые должны принимать комплексные значения, характеризуются атрибутом COMPLEX, как и в фортране. Противоположным по смыслу является атрибут REAL, характеризующий действительные значения в PL/1 и принимаемый по умолчанию (по неявному описанию в фортране). Эти атрибуты называются *атрибутами моды*.

В PL/1 для величин, принимающих комплексные значения, дополнительно задается атрибут FIXED или FLOAT, определяющий тип значений как для действительной, так и для мнимой частей; может быть задан и атрибут точности (см. ниже). Таким образом, для описываемых комплексных величин (переменных, функций) тип значений и точность действительной и мнимой частей всегда совпадают (они могут отличаться только в константах). Приведенные выше константы, если они являются значениями некоторых переменных величин программы, будут иметь, например, следующий вид:

. 01+25I, 4.00—1.56I, —7.00E0—88.8E—4I.

Форма.

число-компл ~
~ {*число-действ* {±} *число-действ-без-зн* I | *число-действ* I}

Точность. Все величины программы, принимающие числовые значения, имеют и такую характеристику, как точность. При описании числовых величин (переменных, функций) точность их представления в машине задается с помощью атрибута точности, который имеет вид одного или двух целых чисел, разделенных запятой и заключенных в скобки. Атрибут точности располагается за одним из арифметических атрибутов, характеризующих тип величины, например:

FIXED(5), FLOAT(7), FIXED(7,5),
COMPLEX FLOAT(10).

Для величин с плавающей точкой точность, задаваемая всегда одним числом, характеризует количество цифр в мантиссе значений

описываемой величины, то есть требуемую точность представления в машине. Например, атрибут точности (7) характеризует значения вида

1.000000E—50, —2222.222E10, 3.141592E0.

Для величин с фиксированной точкой точность определяется одним или двумя числами. В первом случае атрибут точности предназначается для характеристики величин с целыми значениями и задает количество цифр в значении, то есть диапазон изменения описываемой целой величины. Например, атрибут точности (6) характеризует значения в пределах от —999999 до +999999, в частности значения вида

000000, —000004, —008800, 999999.

Во втором случае атрибут точности используется при описании величин, имеющих дробную часть. Первое число в атрибуте, обозначаемое далее через p, определяет общее количество цифр в значении, второе (q) — задает количество цифр в дробной части значения. Например, атрибут точности (7,5) характеризует значения вида

00.00000, 02.00000, —12.34567, +99.99999.

Атрибут точности для величины с фиксированной точкой определяет диапазоны изменения как целой, так и дробной частей значений этой величины. Например, положительные значения в нашем случае будут меняться соответственно в пределах от 00 до 99 и от .00000 до .99999. Напомним, что точность представления величин с фиксированной точкой (как и целых величин) в машине не зависит от приписанного атрибута точности: в отличие от величин с плавающей точкой первые представляются в машине всегда абсолютно точно.

Для комплексных величин атрибут точности определяет точность для обеих частей комплексного значения. Например, атрибуты

COMPLEX FLOAT(10) или COMPLEX FIXED(2)

характеризуют, например, значения вида

8.175432179E—4 +1.500000000E+2I или 15—04I.

Форма.

атрибут-точности ~ (*число-целое* [, *число-целое*])

Замечание 1. В то время как точность представления значений переменных и функций в машине указывается при их описании, точность представления констант, встречающихся в тексте программы, определяется по их внешнему виду и зависит от количества цифр в изображении самого числа или его мантиссы (для плавающей точки). Например, константы

0, —0080, 59.6, 1.54E—4, 1.000E3I.

имеют соответственно следующие атрибуты точности:

(1), (4), (3,1), (3), (4).

Важно отметить, что в ряде случаев точность вычислений в ЭВМ, например, с константами 59.6000 и 59.6000000 (имеющих атрибуты (6,4) и (9,7)) будет в два и три раза выше, чем с аналогичной константой, приведенной выше, что может в дальнейшем сказаться на точности окончательного результата (подробнее об этом см. ниже в 2.1 и в 8.1).

Замечание 2. Реальная точность представления величин в ЭВМ может быть и выше заданной атрибутом точности при ее описании; транслятор гарантирует лишь, что она не будет ниже. Например, особенностью представления в ЕС величин с плавающей точкой является то, что если их точность в PL/1 характеризуется соотношением $p \leqslant 6$, то под их представление отводится 4 байта и они будут иметь реальную точность в 6 знаков; если $p > 6$, то числа занимают 8 байт и точность их представления всегда соответствует 16 знакам (то есть точности p, равной 16). Этой особенностью представления значений с плавающей точкой в ЕС ЭВМ можно воспользоваться для того, чтобы в мантиссе числа задавать меньшее количество цифр. Например, вместо

1.54000E—4 или 1.540000000E—4

можно задать

1.54E—4 или 1.540000E—4

Замечание 3. В фортране ЕС точность (или диапазон — для целых величин) представления величин в машине определяется не количеством цифр в значении величины, а его *длиной*, то есть количеством байт, отводимых для представления значения величины, например:

REAL*8, INTEGER*2, COMPLEX*4

Во многих версиях языка фортран для вещественных величин используются описатели REAL и DOUBLE, характеризующие стандартную или нестандартную (двойную) точность представления данных.

Соответствие между атрибутом точности PL/1 и необходимым количеством байт для представления значения PL/1 в ЕС ЭВМ (длиной — в фортране ЕС) будет приведено ниже, в 8.5.2.

1.7.2. Строчные константы. В PL/1 имеются строчные константы (строки) двух видов: битово-строчные (битовые) и символьно-строчные (символьные). Строки служат для представления в программе последовательности символов или двоичных цифр.

Символьные константы PL/1 представляют собой последовательность символов ЭВМ, заключенную в апострофы, например:

'НОМИНАЛЬНОЕ␣НАПРЯЖЕНИЕ'
'COS(X)' 'ТАБЛИЦА␣1'

Пробел используется в строках наряду с другими символами. Как и в ряде версий фортрана, если апостроф необходимо поместить внутрь строки, то он удваивается. Например, если в программируемых вычислениях нам необходимо оперировать с последовательностью символов вида

КИНОФИЛЬМ␣''ПЕТР␣I''

то в тексте программы следует записать следующую константу:

'КИНОФИЛЬМ␣''''ПЕТР␣I'''''

Аналогичные константы имеются в фортране ЕС и в фортране 77. Литеральные константы фортрана вида 9HТАБЛИЦА␣1 в PL/1 отсутствуют.

Символьные константы PL/1 могут использоваться, в частности, в тех же случаях, что и литеральные константы фортрана, например, при выводе на печать, инициализации или в качестве аргументов.

Форма.

константа-символьная~
~'[*символ-ЭВМ* <кроме апострофа>|'']...'

Как видим, возможна и пустая строка вида ''.

Величины, которые будут принимать символьно-строчные значения, характеризуются атрибутом CHARACTER или сокращенно CHAR. В фортране такие величины отсутствуют, но в ряде случаев допускается присваивание символьных значений (констант) переменным любого типа.

Битовые константы PL/1 соответствуют логическим и шестнадцатеричным константам фортрана и являются их обобщением.

Логические значения **истина** (.TRUE.) и **ложь** (.FALSE.) представляются в тексте программы на PL/1 соответственно битовыми константами '1'B и '0'B. В ходе вычислений в качестве значений этих констант будут взяты двоичные цифры 1 и 0.

Для представления в тексте программы на PL/1 шестнадцатеричных констант, имеющихся в фортране, каждая шестнадцатеричная цифра представляется в двоичном виде, и полученная последовательность двоичных цифр (битов) оформляется в виде битовой строки добавлением апострофов и буквы B. Например, константу фортрана

ZFF70

в PL/1 можно представить так:

'1111111101110000'B

Битовые строки PL/1 имеют гораздо более широкое применение, чем аналогичные константы в фортране.

Форма.

константа-битовая ~ '[0 | 1]...'B

Как видим, допускается и пустая битовая строка вида ''B. Пробелы внутри битовой константы (и перед буквой B) не допускаются.

В PL/1 величинам, которые будут принимать логические (битово-строчные) значения, приписывается атрибут BIT. Можно считать, что в определенном смысле этому атрибуту соответствует описатель LOGICAL в фортране.

Повторитель. Для строчных констант, как для символьных, так и для битовых, допускается сокращенная запись. Например

'ДА — ДА — ДА — ДА —' эквивалентно (4)'ДА —'
'111111111111111'B эквивалентно (15)'1'B

В качестве коэффициента повторения, заключаемого в круглые скобки, могут использоваться только натуральные числа (целые без знака).

Обобщенные формы для строчных констант:

константа-симв ~
~ [(*повторитель*)]'[*символ-ЭВМ* <кроме'> | '']...'
константа-битов ~ [(*повторитель*)]'[0 | 1]...'B
повторитель ~ *число-целое-без-знака*
константа-строчная ~ {*константа-симв* | *константа-битов*}

Длина. Количество символов в символьно-строчном значении или количество битов в битово-строчном называется *длиной* этого значения. Длина для значений строчных величин (переменных, функций) задается при их описании с помощью атрибута длины, который в общем случае имеет вид выражения, заключенного в круглые скобки и располагаемого вслед за атрибутом CHARACTER или BIT. Например:

CHARACTER(10), BIT(2), CHAR(2* K + 1)

Максимальная длина равна 32 767($2^{15}-1$).

Длина константы определяется по количеству символов или битов, содержащихся внутри окаймляющих апострофов, причем для символьных строк каждая пара апострофов внутри строки считается за один. Например, константы

'COS(X)', 'КИНОФИЛЬМ␣''''ПЕТР␣I''''', (4)'ДА —',
'1111111101110000'B, ''B

имеют соответственно атрибуты длины, равные

(6), (20), (12), (16), (0).

В фортране 77 также есть строчные величины и описатель CHARACTER, после которого задается длина в виде

CHARACTER * 5

В отличие от PL/1 длина может быть задана только константой.

1.7.3. Меточные константы. Меточными константами считаются в PL/1 метки, то есть идентификаторы, которые присоединены к какому-либо оператору программы через двоеточие.

1.8. Пробелы и комментарии

Как уже упоминалось ранее, в отличие от фортрана, внутри ключевых слов, идентификаторов (имен), чисел и битовых строк пробелы не допускаются. Пробелы (один или несколько) должны отделять стоящие рядом ключевые слова, идентификаторы, константы (числовые и строчные).

Пробелы могут быть использованы в различных местах программы с целью облегчить чтение составляемой программы при дальнейшей работе с нею. В частности, пробелами удобно выделять знаки операций в выражениях. Кроме того, полезно использовать целые строчки текста программы, состоящие из одних пробелов, для разделения текста на смысловые части; в фортране (не считая фортран 77) пустые строчки текста не допускаются.

Комментарий PL/1 имеет вид последовательности символов ЭВМ, ограниченной парами спецзнаков /* и */ (без пробелов). Например,

/* ЗДЕСЬ ВОЗМОЖНА ПОТЕРЯ ТОЧНОСТИ */

Комментарии могут располагаться и на отдельной строчке текста, как в фортране, и внутри операторов или даже выражений, размещенных на части строчки. Комментарии можно помещать в тех же местах, что и пробелы, например рядом со спецзнаками, константами, идентификаторами и т. п.

Примеры:

M/* ФУНКЦИЯ */ (X, Y)

F/**** ПЕРЕМЕННАЯ ****/ (R, T)

Форма.

комментарий ~ /* *символ-ЭВМ* ⟨кроме сочетания */⟩... */

Упражнения

1.1. Записать на PL/1 следующие конструкции фортрана:

а) TEMA31; б) ☐1; в) ATAN(Z);

г) COTAN(ALPHA); д) (3.5, 15.1);

е) (0.0, 40E—5); ж) 9H␣ПЕЧАТЬ␣' .

1.2. Определить, какие из приведенных идентификаторов PL/1 ются ошибочными:

а) НОМЕР; б) МОПС; в) TASK KOSCHI;

г) A@B; д) C&D; е) S#27.

1.3. Определить, какие из приведенных констант PL/1 являются шибочными:

а) 6.E3; б) .7D—7; в) 55E—003;

г) +.88; д) 4E—2—2I; е) —4.5+I;

ж) '1␣1'B; з) (100)'100'B; и) '1+I';

к) ''—ШТРИХ'; л) (4+5)'1'B; м) '1␣1' .

ГЛАВА 2

ВЫРАЖЕНИЯ

Ориентируясь на фортран, будем сначала различать в PL/1 выражения только двух типов: арифметическое и логическое.

2.1. Арифметическое выражение

Арифметическое выражение PL/1 строится из тех же элементов (переменные, функции, константы, знаки операций, скобки) и по тем же правилам, что и в фортране. Сравните, например, выражения на PL/1 и фортране, реализующие одни и те же вычисления:

```
T/R ** 8 - B(I, J + 1) * 5.5E - 4
```

и

```
T/R ** 8 - B(I, J + 1) * 5.5E - 4
```

Как видим, приведенные выражения совсем не отличаются один от другого.

В большинстве случаев программист, переходя к работе на PL/1, может пользоваться теми навыками, которые он приобрел, программируя выражения на фортране. Однако имеются и некоторые отличия, которые при этом необходимо иметь в виду.

2.1.1. Старшинство операций. Старшинство арифметических операций в PL/1 то же, что и в фортране. Несколько подряд идущих операций возведения в степень выполняются справа налево, как и в фортране ЕС, а не слева направо, как во многих версиях фортрана; остальные операции в рассматриваемом случае выполняются слева направо. Например, последовательность вычисления выражения

```
1 - X - Y ** Z ** T
```

будет следующей:

```
1 - X, Z ** T, Y ** (Z ** T), (1 - X) - (Y ** Z ** T)
```

2.1.2. Тип результата. В PL/1 нет никаких ограничений на использование в выражении арифметических величин различных типов

(целый, плавающий, комплексный); в частности, в отличие от фортрана, допускается комплексная величина и в качестве показателя степени. Таким образом, операнды любой операции могут принадлежать к разным типам, но, как и в фортране, если операнды принадлежат к одному типу, то вычисления производятся значительно быстрее.

Когда операнды какой-либо операции имеют разные типы, то один из них перед выполнением операции приводится к типу, являющемуся «старшим». Старшинство типов данных в PL/1 то же, что и в фортране:

комплексный ← плавающий ← фиксированный
(вещественный) (целый)

Таким образом, если один из операндов с плавающей точкой, то и результат этой операции плавающий; если один из операндов комплексный, то и результат комплексный. Напомним, что в PL/1 комплексный результат может быть или в плавающей, или в фиксированной форме, которая определяется по тому же правилу старшинства типов данных. То есть в PL/1, на самом деле, используются следующие два вида преобразований:

комплексный ← действительный
плавающий ← фиксированный

Первое преобразование заключается в добавлении нулевой мнимой части, и второе — в выделении порядка и мантиссы. Например:

1.23456E + 2 — (13 + 481) ⟶ 1.10456E + 2 — 4.80000E + 11

Заметим здесь, что значения с фиксированной точкой после преобразования к плавающей форме будут уже представляться не абсолютно точно, а с ограниченной точностью, зависящей от количества цифр в этих значениях.

В случае, когда оба операнда целые, то в PL/1 результат не всегда получается целым. Результат операции деления, в отличие от деления в фортране, может получиться и дробным, то есть имеющим тип с фиксированной точкой. Например, в PL/1 получим 5/2 ⟶ 2.5, а в фортране 5/2 ⟶ 2. Впрочем, в этом нет ничего удивительного, если вспомнить, что в PL/1 целый тип является частным случаем типа данных с фиксированной точкой.

Точное правило для определения типа результата операции возведения в степень приведено позже (см. 8.1.1), а пока ради простоты будем считать, что результат всегда имеет тип плавающий (вещественный), что в подавляющем большинстве случаев соответствует действительности.

Если почему-либо необходимо, чтобы тип результата операции деления или возведения в степень в PL/1 соответствовал установ-

ленному в фортране, то следует применить функцию FIXED, приводящую тип полученного результата к целому типу. Например, осуществляется следующая замена выражений фортрана на выражения PL/1 (I, J — целые):

I/J ⟶ FIXED(I/J)

и

I ** 3 ⟶ FIXED(I ** 3 ± 0.5)

В обоих случаях, однако, приведенное соответствие справедливо лишь для случая, когда аргумент по модулю меньше 10^5; знак перед 0,5 соответствует знаку I (можно использовать и функцию SIGN — см. 9.1.2).

2.1.3. Точность результата. Точность результата операции для плавающей формы, как и в фортране, определяется по точности операндов и берется равной большей из точностей двух операндов (для фортрана имеется в виду длина). Например, если принять, что переменные P, Q, R имеют соответственно атрибуты точности, равные (5), (7), (9), то результаты вычисления выражений

P+Q, Q * R, R — Q/3

будут иметь точность, соответствующую следующим атрибутам:

(7), (9), (9).

Точность результата большинства операций в форме с фиксированной точкой является абсолютной, как и точность операндов. Но для результата операции деления, который может быть и неточным, берется максимальная точность ($p = 15$). Некоторые особенности выполнения операций над операндами с фиксированной точкой будут приведены позднее (см. 8.1).

Здесь лишь отметим, что точность вычисления стандартных функций, упомянутых ранее (см. 1.6), жестко соответствует точности заданного аргумента. Поэтому, например, значение указателя функции LOG10(5) вычисляется всего лишь с одним знаком точности и будет равно 0.6E0, вместе с тем

LOG10(5.0000) = 0.6990E0 и LOG10(00050) = 1.6990E0

Таким образом, программист должен особенно внимательно следить за теми указателями функций, аргументами которых являются числа, и в необходимых случаях увеличивать их точность, приписывая к ним нули (слева или справа).

2.1.4. Переполнения. В PL/1, как и в фортране, нужно следить за тем, чтобы результат выполнения каждой операции не выходил за установленный для данной машины предел. В фортране ЕС для целых значений таким пределом является 2^{31} ($\approx 2 \cdot 10^9$), а для вещественных — 10^{75}. Мы будем пока считать, что в PL/1 приняты

те же пределы, хотя для целых результатов в ряде случаев, этот предел оказывается равным 10^{15}.

Кроме того, при выполнении программ, написанных на PL/1, может возникнуть так называемое «фиксированное» переполнение, в том случае, если получается результат в форме с фиксированной точкой (то есть с дробной частью без порядка), в котором оказывается более 15 цифр. Подробно о мерах предупреждения этого переполнения будет рассказано в 8.1.1.

2.2. Логическое выражение

В соответствии с терминологией, принятой для строчных констант, логическое выражение в PL/1 правильнее называть *битово-строчным* или *битовым* выражением. Логическое или битовое выражение PL/1 строится из тех же компонент (логические или битовые константы, переменные, функции, операции) и, в основном, по тем же правилам, что и в фортране. Только операции и константы в PL/1 имеют другой вид. Сравните

X .LE. 1 и X <= 1

X * Y .GT. 2 .OR. .NOT. (L .AND. .TRUE.)

и X * Y > 2 ! ¬(L & '1'B)

2.2.1. Битовые операции. В приведенной ниже таблице дано соответствие ряда операций фортрана битовым операциям PL/1, то есть операциям, результаты которых представляются битовыми строками.

фортран	.GT.	.GE.	.LT.	.LE.	.EQ.	.NE.	.NOT.	.A D.	.OR.
PL/1	>	>=	<	<=	=	¬=	¬		!

Между составными символами операций (>=, =<, ¬=) пробелы не допускаются.

Старшинство операций то же, что и в фортране, за одним исключением: одноместная операция отрицания является операцией самого старшего ранга, того же самого, что и операция возведения в степень. Поэтому выражение, которое в фортране имеет вид

.NOT. X .GE. 2

в PL/1 должно быть записано так:

¬(X >= 2)

Битовые операции разделяются на операции отношения и логические операции. Другие действия, выполняемые над битовыми величинами, описываются позже (см. 8.2).

Замечание. В PL/1 нет логических операций, эквивалентных операциям .EQV. и .NEQV., имеющихся в фортране 77, но они вполне заменяются операциями = и ¬= над битовыми строками.

2.2.2. Операции отношения. В PL/1 можно сравнивать между собой как арифметические значения, так и строчные (битовые и символьные).

Сравниваемые арифметические значения, имеющие разные атрибуты, как и при выполнении арифметических операций, приводятся к старшим атрибутам (см. 2.1). Допускается и сравнение комплексных выражений; но, разумеется, только с помощью операций = и ¬=, например,

```
X ¬= 1+1I
```

Сравнение строчных значений (битовых или символьных) производится побитно (посимвольно) слева направо до первого несовпавшего бита (символа). Та строка, которой принадлежит бит (символ), имеющий большее значение, считается большей (или неравной другой); если все соответствующие биты (символы) равны между собой, то строки считаются равными. Сравнение символов производится на основании принятой кодировки, которая приведена в 9.6. Когда одна из сравниваемых строк короче другой, то перед сравнением производится дополнение справа более короткой строки нулями (для битовой строки) или пробелами (для символьной строки).

Результатом любой операции отношения являются значения **истина** или **ложь**, представляемые в машине двоичными единицей или нулем и изображаемые битовыми константами '1'B и '0'B.

Ниже даются примеры операций отношения, результаты которых имеют значение **истина** ('1'B):

```
'1'B > '0'B     '1'B = '100'B   '11000'B > '10101'B
'011'B < '1'B   '—' > '+'       '5—21' > '5+21'
'КОН' < 'КОНЕЦ'
```

Замечание. Помимо перечисленных выше в PL/1 имеются еще операции ¬> и ¬<, эквивалентные операциям <= и >=.

2.2.3. Логические операции. В PL/1 логические операции &, |, ¬ (**и**, **или**, **не**) могут производиться не только над элементарными битовыми значениями, представляемыми константами '1'B и '0'B, но и над совокупностью битов, то есть битовыми строками произвольной длины. Если битовые строки имеют разные длины, то они предварительно выравниваются путем добавления недостающих двоичных нулей справа к более короткой строке. Логическая операция производится побитно над соответствующими битами обеих строк. Длина результата равна длине более длинной строки.

Примеры.

¬'10110'B ⟶ '01001'B

'1110011'B & '1'B ⟶ '1000000'B

'10101'B ! '0011'B ⟶ '10111'B

$$X >= 1 \,\&\, X <= 2 \longrightarrow \begin{cases} \text{'1'B для } x \in [1,2] \\ \text{'0'B для } x \notin [1,2] \end{cases}$$

Замечание. Значение логического выражения PL/1, содержащего только битовые нули, в операторе IF (см. 3.5) эквивалентно константе '0'B (ложь); если в значении содержится хотя бы одна битовая единица, то оно эквивалентно константе '1'B (истина).

2.3. Старшинство операций

Старшинство операций в PL/1 установлено в соответствии со следующей сводной таблицей операций (операция !! описана в 8.2.2).

1. **, ¬ (возведение в степень, НЕ—логическое отрицание).
2. *, / (умножение, деление).
3. +, — (сложение, вычитание).
4. !! (сцепление).
5. <, <=, >, >=, =, ¬=, ¬>, ¬< (сравнения).
6. & (И—логическое умножение).
7. ! (ИЛИ—логическое сложение).

Напомним, что в фортране логическое отрицание по старшинству стоит после операций сравнения; операция сцепления в фортране 77 имеет то же старшинство, что и в PL/1.

Упражнения

2.1. Определить точность следующих выражений PL/1 (L—целое точности 5, X и Z—с плавающей точкой точности соответственно 8 и 10):

а) 1/EXP(1.54)+6.0

б) 1/EXP(1.54 * X)+6.0 * L

в) 1/EXP(1.54 * X)—6.0 * L * Z

2.2. Написать на PL/1 следующие выражения:

а) $$\frac{R_i + R_{i+1}}{C_1 \cdot C_2} + \frac{a^k - \dfrac{x}{c} + 0.5}{\dfrac{1}{2} + \dfrac{x^3}{R_{i-1} - R_i}};$$

б) $$\frac{R_1}{R_2} \cdot \frac{\left(\sqrt{\beta} - e^{3 \cdot 10^{-5} \cdot x^{0.3}} + R_2 \cos\left(\sqrt[3]{\pi y}\right)\right)^{x^{1/7}}}{2(1-\hat{R})}.$$

2.3. Написать на PL/1 следующие выражения фортрана:

а) X * Y .NE. 0 .AND. .NOT. B

б) X .LT. 1 .OR. .NOT. Y .EQ. Z .AND. Z .NE. X

2.4. Определить истинность следующих выражений (см. таблицу кодировки в 9.7):

а) '1100'B¬='11'B
б) ''B < '0'B
в) '011'B < '10'B
г) ('1110'B & '01'B) > '001'B
д) ('0011'B ! '1'B)<'101'B
е) 'BIT' < 'BIG'
ж) 'BIT' < 'BITOW'
з) 'BIT' < 'BIT '
и) 'BIT' < 'SYMBOL'
к) 'БИТ' > 'СИМВОЛ'
л) 'ИВАНОВ' > 'ИВАШКЕВИЧ'

ГЛАВА 3

ОПЕРАТОРЫ

Здесь мы рассмотрим, в основном, только те операторы PL/1, которые имеют аналог среди операторов фортрана. Каждый оператор PL/1 имеет следующую общую форму:

[метка:] ... тело-оператора;

Каждый оператор может быть помечен, причем даже несколькими метками. Ниже при изложении операторов синтаксические формы, как правило, будут даваться для непомеченных операторов. Кончается оператор PL/1 всегда точкой с запятой.

В отличие от фортрана в одной строчке бланка PL/1 может быть расположено произвольное число операторов или любая часть оператора. Операторы располагаются в произвольных позициях рабочего поля бланка, занимающего позиции со 2-й по 72-ю.

3.1. Оператор присваивания

Оператор присваивания PL/1, в основном, имеет тот же вид, что и в фортране, если не считать ограничивающей его точки с запятой. Сравните:

```
      A=B+1;          A=B+1
LOB: G(I)=Z<14;   15 G(I)=Z.LT.14
```

Форма.

оператор-присваивания~переменная=выражение;

Оператор присваивания PL/1 может быть четырех типов: арифметического, битового, символьного и меточного. Тип оператора зависит от типа переменной и выражения, находящихся в левой и правой частях оператора присваивания.

3.1.1. Арифметическое присваивание. В случае, если арифметические переменная и выражение в левой и правой частях оператора имеют различные атрибуты, то так же, как и в фортране,

автоматически производится приведение атрибутов выражения к атрибутам переменной, стоящей в левой части оператора.

Приведение плавающего значения к целому производится отбрасыванием дробной части. Комплексное значение, как и в фортране, приводится к вещественному отбрасыванием мнимой части. Приведение целого значения к плавающему виду и последнего к комплексному осуществляется по тем же правилам, что и при вычислении арифметических выражений, когда операнды операций имеют разные атрибуты. Например, если обозначить целую, плавающую и комплексную переменные соответственно через I, R и C, то преобразования между ними, вызываемые соответствующими операторами, могут иметь следующий вид:

I = R;	R = I;
$-13\overset{4}{\underset{3}{\leftarrow}}-1.3699E1$	$1.3E1\underset{4}{\overset{3}{\leftarrow}}13$
R = C;	C = R;
$1.4E3\underset{4}{\overset{3}{\leftarrow}}1.4E3+3.8E-2I$	$1.4E3+0.0E0I\underset{4}{\overset{3}{\leftarrow}}1.4E3$

Если значения левой и правой частей оператора имеют разную точность, то происходит приведение точности выражения к точности переменной в левой части. Для случая переменной с плавающей точкой в левой части оператора присваивания отсекаются цифры или добавляются нули в младшие разряды значения; для целой переменной — в старшие; для фиксированной точки — может быть, и в младшие и в старшие разряды. Для случая целой или фиксированной переменной, когда при присваивании теряются старшие значащие разряды значения, фиксируется ошибка типа SIZE (диапазон); если не принять специальных мер (см. 7.3), то выполнение программы будет продолжаться с неправильным результатом присваивания (что может произойти и в фортране ЕС при присваивании целым переменным длины 2 целых значений, превышающих по абсолютной величине 2^{15}).

Ниже даны примеры приведения значений 01150 и 1.2349E1, имеющих атрибут точности (5), к другим атрибутам точности: к (7), (4) и (3).

Для целых	Для плавающих
0001150 ← 01150	−1.234900E1 ← −1.2349E1
1150 ← 01150	−1.234E1 ← −1.2349E1
150 ← 01150 (ошибка)	−1.23E1 ← −1.2349E1

Для значений с фиксированной точкой (сверху указаны атрибуты точности):

(7,4)	(5,3)	(3,1)	(5,3)	(3,2)	(5,3)	
013.8600 ←	13.860	13.8 ←	13.860	3.86 ←	13.860	(ошибка)

3.1.2. Строчные присваивания. Для битово-строчного присваивания в левой части оператора присваивания должна находиться битовая переменная, а в правой части — битовое выражение. Если длина значения битового выражения не совпадает с длиной переменной, то происходит приведение длины: от значения выражения с п р а в а отбрасываются лишние биты или приписывается необходимое количество нулей.

При символьно-строчном присваивании в случае, когда длины переменной и выражения не равны, от значения выражения справа отбрасываются лишние символы или к нему приписывается необходимое количество пробелов (как и в фортране 77).

П р и м е р ы.

Для строчных переменных T и H, характеризуемых соответственно атрибутами BIT(6) и CHARACTER(5), в результате выполнения приведенных ниже операторов присваивания получим следующие значения (см. справа):

оператор	результат
T='1010'B;	T='101000'B
T=¬'11011011'B;	T='001001'B
H='ОТБРОСИТЬ';	H='ОТБРО'
H='';	H='␣␣␣␣␣'

3.1.3. Меточное присваивание. Оператор меточного присваивания служит для присваивания меточной переменной значения другой меточной переменной или метки. Такой оператор соответствует оператору ASSIGN в фортране.

Например, оператор фортрана

ASSIGN 12 TO N

в PL/1 может быть записан следующим образом (если предположить, что метке 12 соответствует в PL/1 метка L12):

M=L12;

Переменная M в программе на PL/1 должна быть описана (см. гл. 4) с атрибутом LABEL, например, оператором

DECLARE M LABEL;

Возможен также и следующий оператор:

BRANCH=M;

если переменная BRANCH также меточная.

Меточные переменные используются в операторе GO TO (см. ниже 3.2).

3.1.4. Список левой части. Одним оператором присваивания в PL/1 можно изменить значения сразу нескольких переменных, поместив их в левую часть оператора присваивания и разделив при

этом запятыми. Например:

P, IND, A(L, K+1) = 0;

Для арифметического присваивания в список левой части могут входить переменные с различающимися атрибутами (FIXED, FLOAT, COMPLEX); переменные могут иметь и разную точность. Значения индексов вычисляются до вычисления выражения правой части. Вычисленное значение правой части по очереди слева направо присваивается всем переменным левой части, и при этом выполняются необходимые преобразования присваиваемого значения.

3.1.5. Формы.

оператор-присваивания ~

~ { *переменная-арифм*,.. = *выражение-арифм*;
переменная-битов,.. = *выражение-битов*;
переменная-симв,.. = *выражение-симв*;
переменная-меточн,.. = {*метка* | *переменная-меточн*}; }

3.2. Оператор перехода

Оператор перехода в PL/1 соответствует оператору GO TO в фортране и имеет следующую форму:

оператор-перехода ~ GO TO { *метка* | *переменная-меточная* };

В случае, когда в операторе используется метка (в терминологии PL/1 — *меточная константа*), то это эквивалентно безусловному оператору GO TO в фортране. Например, оператору фортрана

GO TO 150

может соответствовать следующий оператор в PL/1:

GO TO L150; или GO TO WWOD;

Меточная переменная используется в операторе перехода PL/1 обычно в тех же случаях, когда при программировании на фортране используется назначаемый или вычисляемый оператор GO TO.

Назначаемый оператор фортрана

GO TO M, (35, 44, 12)

реализуется, например, следующими операторами в PL/1:

```
DECLARE M LABEL(L35, L44, L12);
  GO TO M;
```

Метки L35, L44, L12 и т. п. выбраны с целью указания на их соответствие меткам фортрана; в реальных программах на PL/1 они обычно берутся более мнемоничными.

В операторе описания, который подробно разбирается в гл. 4, описывается меточная переменная M и указываются те метки, ко-

торые могут являться значениями описываемой меточной переменной; в фортране соответствующий список меток задается в самом операторе GO TO. Если перед выполнением оператора

GO TO M;

был выполнен оператор меточного присваивания

M = L12;

то переход будет осуществлен на оператор с меткой L12. Если метка L12 отсутствует в списке, то при выполнении оператора GO TO M зафиксируется ошибка, и выполнение программы прекратится.

Отличием назначаемого оператора GO TO от соответствующего оператора перехода в PL/1 является то, что в PL/1 список меток, контролирующий переход, задается раз и навсегда при описании меточной переменной, а в фортране каждый раз в выполняемом операторе перехода.

Замечание. В силу отмеченного различия при реализации формируемого (изменяемого) оператора перехода в PL/1 и фортране, если сопоставлять между собой соответствующие программы на этих языках, реализующие один и тот же алгоритм, то может оказаться, например, что в этих программах потребуется различное количество меточных переменных.

Вычисляемый оператор GO TO фортрана

GO TO (21, 42, 25, 34), I

реализуется в PL/1, например, следующими операторами:

```
DECLARE P(4) LABEL INITIAL(L21, L42, L25, L34);
  GO TO P(I);
```

Величина P описывается как меточный массив (вектор), состоящий из 4 элементов, которым с помощью атрибута начальных значений (см. ниже 4.3) присваиваются указанные в списке меточные значения. Конструкция P(I) является переменной с индексом. Переменная I, как и в фортране, предполагается целого типа (или приводимой к нему) и должна быть соответствующим образом описана. Если перед выполнением оператора перехода переменная I получила, например, значение 3, то следующим будет выполняться оператор, помеченный меткой L25. Если значение I не удовлетворяет соотношению $1 \leqslant I \leqslant 4$, то будет зафиксирована ошибка при выполнении программы. В PL/1, как и в фортране 77, вместо I может быть задано и выражение.

Вслед за атрибутом LABEL, как и выше, может быть задан контрольный список меток, но здесь он обычно является лишним, так как совпадает со списком, задаваемым в атрибуте INITIAL.

Значения элементов меточного массива (см. выше P), установленные при описании, могут быть изменены в ходе выполнения программы с помощью меточных операторов присваивания. Например, если перед выполнением оператора перехода по меточной переменной P(I) был выполнен оператор

```
P(3) = WWOD;
```

то для I=3 оператор

```
GO TO P(I);
```

осуществляет переход на метку WWOD, а не на L25, как раньше. Если программист использует такие переприсваивания в своей программе, то следует для контроля за ними задать в операторе DECLARE контролирующий список меток вслед за атрибутом LABEL, то есть написать, например,

```
LABEL(L21, L42, L25, L34, WWOD)
```

Нужно заметить, что следует очень осторожно пользоваться операторами перехода по меточным переменным (особенно аналогичных назначаемому оператору фортрана), так как они ведут к чрезвычайно трудно находимым ошибкам при отладке программ. Хотя можно заметить, что оператор перехода PL/1 и проигрывает несколько по своим возможностям оператору фортрана, но этот проигрыш только содействует тому, чтобы воспрепятствовать созданию на PL/1 слишком запутанных по своей логике программ.

3.3. Оператор вызова

Оператор вызова (CALL), как и в фортране, служит для обращения к подпрограмме (в PL/1 — процедуре) с заданием ей необходимых аргументов. Сравните:

```
CALL SUM(A,B,C); и CALL SUM(A, B, C)
```

Особенности задания аргументов и выполнения процедур обсуждаются в гл. 5 вместе с правилами описания процедур.

Форма.

оператор-вызова ~ CALL *имя-процедуры* [(*аргумент*,..)];

3.4. Пустой оператор

Пустой оператор в PL/1 соответствует оператору CONTINUE в фортране. Он используется, в основном, для помещения метки в нужное место программы. Например, оператору

```
240 CONTINUE
```

в PL/1 может соответствовать оператор, имеющий вид

```
L240: ; или OBHOD: /* ПУСТОЙ */;
```

Непомеченный пустой оператор содержит только точку с запятой.

Форма.

оператор-пустой ~ <пусто>;

3.5. Условный оператор

Условный оператор PL/1 имеет следующую форму:

оператор-условный ~

~ IF *выражение-логич* THEN *оператор*-1 [ELSE *оператор*-2]

Точки с запятой в форме явно не показаны, но ими кончаются оба оператора, входящие в условный; «своей» точки с запятой условный оператор не имеет.

Условный оператор PL/1 без компоненты ELSE имеет прямую аналогию с логическим условным оператором фортрана. Сравните:

IF (X .LT. 0) X = — X и IF(X < 0) THEN X = — X;

Как следует из приведенной формы, заключение логического выражения в скобки для PL/1 является не обязательным.

Если в условном операторе имеется компонента с ELSE, то оператор, стоящий за этим ключевым словом, выполняется в том случае, когда логическое выражение условного оператора имеет значение ложь. Например, после выполнения оператора

IF(X < Y) THEN M = Y; ELSE M = X;

переменная M получит значение max(x, y), так как при X < Y выполняется оператор M = Y;, а при X ≥ Y — оператор M = X;. В обоих случаях вслед за условным оператором будет выполняться оператор, расположенный непосредственно за оператором M = X; (если, конечно, входящие операторы не являются операторами перехода)

На фортране этот пример выглядит следующим образом:

```
      IF    (X .GE. Y) GO TO 15
            M = Y
        GO  TO 20
15      M = X
20 . . . . . .
```

В качестве операторов, входящих в условный, в PL/1 могут использоваться любые операторы, кроме оператора описания. В частности, каждый из входящих операторов может, в свою очередь, быть условным оператором, то есть допустим, например, следующий оператор

```
IF X < 0 THEN S = — 1;
         ELSE IF X > 0 THEN S = 1;
                       ELSE S = 0;
```

реализующий функцию sign (x):

$$s = \begin{cases} 1, & \text{при } x > 0; \\ 0, & \text{при } x = 0; \\ -1, & \text{при } x < 0. \end{cases}$$

Для вложенных условных операторов, в случае когда некоторые его компоненты с ELSE отсутствуют, встает задача по определению соответствующих THEN и ELSE. Пары THEN—ELSE устанавливаются при просмотре условного оператора справа налево; если при этом встречаются подряд две конструкции с THEN, то считается, что для левого THEN соответствующего ELSE нет. Например, в нижеследующем условном операторе соответствующие THEN и ELSE для наглядности расположены одно под другим или на одной строчке:

```
IF M > 0 THEN
   IF N > 0    THEN IF L=0 THEN L=1; ELSE;
               ELSE K=1;
```

Для первого THEN соответствующего ELSE нет. Если опустить ELSE;, содержащее пустой оператор, то последнее ELSE стало бы относиться к THEN с L=1;.

Следует весьма осторожно прибегать к вложенным операторам, особенно когда в них отсутствуют конструкции с ELSE.

Напомним, что и после THEN и после ELSE можно поместить лишь один оператор PL/1, и поэтому следующая конструкция является ошибочной:

```
IF X > 1 THEN C=C+1; X=X*10; ELSE C=-1; X=X/10;
```

Используя составной оператор (см. ниже 3.6), можно обойти это ограничение.

Замечание. Оператор с аналогичными возможностями имеется и в фортране 77, но там каждый условный оператор, в том числе и вложенный, ограничивается ключевым словом ENDIF.

Арифметический условный оператор фортрана, в том случае, когда из трех используемых в операторе меток две совпадают, сводится к одному условному оператору PL/1. Например, оператору фортрана

```
IF(Y) 14, 15, 15
```

в PL/1 соответствует оператор

```
IF(Y < 0) THEN GO TO L14;
          ELSE GO TO L15;
```

В том случае, когда все три метки в условном арифметическом операторе фортрана различны, такому оператору в PL/1 соответствует вложенный («гнездованный») условный оператор. На фортране

программа для вычисления функции sign (см. выше) выглядит так:

```
      IF(X) 50, 60, 70
50      S=-1
      GO TO 100
60      S=0
      GO TO 100
70      S=+1
100   . . . . . .
```

Замечание. Операторы, входящие в условный оператор, могут быть помечены, и на них может быть осуществлен переход извне условного оператора. В этом случае следующим оператором всегда будет выполняться оператор, стоящий за условным (если не был осуществлен выход из условного оператора по оператору перехода).

3.6. Составной оператор

Составной (групповой) оператор используется в тех случаях, когда несколько операторов требуется объединить в один. Это осуществляется с помощью «операторных» скобок — операторов DO; и END;. Составной оператор обычно используется внутри условного оператора, например (ср. с примером в 3.5):

```
IF A < B THEN DO; A=A+1; GO TO S1; END;
IF X > 1 THEN DO; C=C+1; X=X*10; END;
         ELSE DO; C=C-1; X=X/10, END;
```

Форма.

оператор-составной ~ DO; *оператор*... END;

Операторы внутри составного оператора могут быть помечены, и на них возможен переход извне.

3.7. Оператор цикла

Читателю, который при программировании на фортране следует хорошему правилу всегда ставить в конце цикла оператор CONTINUE, будет легко установить соответствие между циклами на фортране и PL/1. Сравните:

```
      DO 50 I=2, N, 1      DO I=2 TO N BY 1;
        S=S+A(I)             S=S+A(I);
50    CONTINUE             END;
```

В PL/1 END-оператор в конце цикла является обязательным — он ограничивает действие оператора DO, называемого оператором *заголовка цикла* и составляет с ним пару. Оператором цикла будем

называть все операторы от DO до END. Обратите внимание на наличие точки с запятой в конце этих операторов.

Как в фортране, параметр шага в операторе DO может отсутствовать, в этом случае он берется равным единице.

В PL/1 для оператора цикла и его заголовка сняты многие ограничения, имеющиеся в фортране. Переменная в заголовке цикла, управляющая его выполнением — *параметр цикла*, может быть не только целого типа, но и плавающего; она может иметь индексы. В качестве начального и конечного значения параметра цикла, а также шага его изменения могут быть заданы любые арифметические выражения, принимающие целые (фиксированные) или плавающие значения, как и в фортране 77. Например, возможны следующие заголовки цикла PL/1:

```
DO R=0 TO T**V BY 1E-1;
```

или

```
DO X=X0 TO XKOH BY (XKOH-X0)/(M-1);
```

Существенным отличием цикла PL/1 является то, что проверка на окончание цикла в PL/1 производится до выполнения операторов, составляющих его тело, а в фортране, как правило, — после. Это справедливо и для первого выполнения тела цикла, поэтому, если перед выполнением приведенных заголовков цикла $N < 2$ (см. выше) или $T**V < 0$, то циклы в PL/1 не будут выполняться ни разу; в фортране при $N < 2$ цикл выполняется один раз.

В операторе цикла PL/1 (как и в фортране 77) допускается задание и отрицательного шага приращения для параметра цикла. В этом случае окончание цикла происходит в тот момент, когда текущее значение параметра цикла становится меньше заданного после TO конечного значения. Например, если в приведенном заголовке заданы такие X0 и XKOH, что $X0 > XKOH$, а $M > 1$, то поскольку шаг, заданный после BY, имеет отрицательное значение, цикл будет выполняться пока $X \geqslant XKOH$.

Как и в фортране, значения границ параметра цикла и его шага, заданные в заголовке цикла, не могут быть изменены в теле цикла. Но значения переменных, входящих в выражения, управляющие изменением параметра цикла, можно изменять, например, с помощью операторов присваивания. Если в приведенном выше примере перед входом в цикл N имеет значение 10, а в теле цикла имеется еще и оператор N=N*2; то величина N каждый раз будет удваивать свое значение, но заданное в заголовке цикла конечное значение параметра цикла останется неизменным, по-прежнему равным 10.

При выходе из цикла PL/1 значение переменной, являющейся параметром цикла, всегда определено, в отличие от фортрана, и равно тому значению, которое в первый раз выходит за заданное

в заголовке конечное значение. В приведенном выше первом примере после окончания работы цикла переменная I будет иметь значение N+1 (N—целое). Для параметра цикла с плавающей точкой на его вычисляемое значение могут влиять ошибки округления при получении нового значения параметра цикла. Например, конечное значение переменной R после выхода из цикла может лежать в интервале (T**V, T**V+0.1).

Запрещается разрешенный в фортране возврат во внутрь цикла после выхода из него по GO TO (так же, как и в фортране 77). Но допускается возврат в цикл из подпрограммы, в том числе и по метке. Тела циклов, как и в фортране, не могут пересекаться частично; одно из них (при пересечении) должно целиком содержаться в другом.

Если оператор заголовка цикла помечен, то в операторе END, ограничивающем тело цикла, может быть задана эта метка, что для циклов больших объемов позволяет сделать запись программы более наглядной. Например, если оператор цикла, приведенный ранее, имеет метку SCAL, то она может быть указана и в END-операторе:

```
END SCAL;
```

Это замечание справедливо и для составного оператора (см. 3.6).

Ф о р м а.

оператор-END ~ END[*метка*];

оператор-цикла ~

~ [*метка*:] ... *заголовок-цикла оператор*... *оператор*-END

заголовок-цикла ~

~ DO *перем*-A = *выраж*-A TO *выраж*-A [BY *выраж*-A];

Здесь *выраж*-A обозначает арифметическое выражение.

Другие возможности оператора цикла PL/1 обсуждаются в 3.8.

3.8. Оператор цикла. Дополнение

Оператор цикла в PL/1 по сравнению с оператором фортран имеет дополнительные возможности, которые перечисляются ниже.

Заголовок цикла, описанный ранее (см. п. 3.7), может быть снабжен ключевым словом WHILE («пока»), задающим дополнительное условие продолжения цикла. В том случае, если перед очередным выполнением тела цикла условие, заданное в скобках вслед за этим ключевым словом, ложно, то цикл прекращает свою работу. Например, заголовок цикла вида

```
DO I=1 TO 40 BY 1 WHILE(S<0);
```

задает кратность цикла, равную 40, но с условием, что при этом будет все время выполняться S<0. Окончание цикла произойдет в случае, когда окажется, что I>40 или S ≥ 0.

В заголовке цикла может отсутствовать ключевое слово TO, управляющее окончанием цикла, и в этом случае окончание цикла должно быть осуществлено по условию, заданному после WHILE. Например, для заголовка цикла

DO I=1 BY 1 WHILE(A(I) ¬=0);

окончание цикла произойдет в тот момент, когда очередной элемент массива A окажется равным нулю (если, конечно, исключить из рассмотрения возможность более раннего выхода из цикла по оператору GO TO).

Условие, заданное после WHILE, проверяется и перед первым выполнением тела цикла; в случае, если оно ложно, тело цикла не будет выполнено ни разу.

Нужно предупредить, что если в последнем приведенном заголовке цикла опустить ключевое слово BY, то заголовок цикла будет задавать не многократное выполнение цикла с шагом, равным единице, а только однократное. Например, для заголовка цикла вида

DO I=1 WHILE(A(I) ¬=0);

тело цикла будет выполнено один раз (для I=1) или не выполнено ни разу, в случае, если окажется, что A(1)=0.

Таким образом, многократное выполнение тела цикла определяется наличием ключевых слов TO или BY в заголовке цикла; в противном случае, если отсутствуют оба эти ключевые слова, то считается, что задано однократное выполнение цикла. Во всех случаях (с BY и TO, с TO, с BY, без BY и TO) тело цикла может быть не выполнено ни разу, если уже перед первым выполнением тела цикла условие, заданное после WHILE, является ложным.

Возможен заголовок и без WHILE вида

DO I=1 BY 1; или DO I=1;

В первом случае задается бесконечный цикл, во втором — однократный. Выход из тела цикла для первого заголовка должен быть осуществлен по оператору GO TO. Вторым заголовком задается однократное выполнение тела цикла и применяется такая форма, как правило, только в операторе со *списком цикла,* имеющим, например, следующий вид:

DO I=1, 3, 5, 6, 7, 8, N—1, N; A(I)=0; END;

Оператор, составляющий тело цикла, будет выполнен многократно для значений параметра цикла, заданных в элементах списка цикла и разделенных запятыми. В общем случае, эти элементы могут быть заданы не только константами и выражениями, но и конструкциями, содержащими ключевые слова BY, TO, WHILE (имею-

щими смысл, описанный выше). Например:

```
DO I=1 TO 5 BY 2,6 TO 8 BY 1 WHILE(S < 0), N—1, N;
    S=S*A(I);
END;
```

Данный оператор может быть представлен в виде, например, трех операторов цикла с одним и тем же телом и с заголовками вида

```
DO I=1 TO 5 BY 2;
DO I=6 TO 8 BY 1 WHILE(S < 0);
DO I=N—1, N;
```

Приведенные возможности оператора цикла PL/1 позволяют программисту всю организацию циклического процесса сосредоточить в его заголовке, а не рассредоточивать управление циклом среди операторов его тела. Это помогает сделать запись операторов цикла гораздо более наглядной. В PL/1, в частности, может быть успешно выполнено одно из требований структурного программирования, запрещающего выход из тела цикла по GO TO.

Все, что говорилось до сих пор, относится к заголовку цикла *с параметром цикла*. В PL/1 возможен и другой тип заголовка цикла—*без параметра цикла*. Например, тело следующего оператора цикла:

```
DO WHILE(X >=1); X= X/10; Y=Y+1; END;
```

будет выполняться до тех пор, пока $X \geqslant 1$; если уже при входе в цикл $X < 1$, то тело цикла не будет выполнено ни разу. Список цикла для заголовка без параметра цикла невозможен.

Обобщая все вышесказанное, можно дать следующие синтаксические формы для оператора цикла:

оператор-цикла ~
~*заголовок-цикла тело-цикла конец-цикла*~
~ DO { WHILE (*выражение-логическое*) / *параметр-цикла=список-цикла* } ;
оператор ... END-*оператор*

список-цикла~
~{*выраж-арифм* [TO *выр-арифм*] [BY *выр-арифм*]
[WHILE (*выр-логич*)]} , ..

параметр-цикла~*переменная*

Ключевые слова BY и TO вместе с относящимися к ним выражениями могут быть переставлены местами.

Оператор цикла и описанный ранее составной оператор являются в PL/1 частным случаем оператора *группы*, который можно представить только что приведенной формой, но скобки { и } нужно

заменить на [и], что позволит выразить и форму для составного оператора (ср. с 3.6).

Пример. Дан вектор a_i ($i=1, 2, \ldots, 30$). Найти сумму элементов до первого, равног нулю, и сумму остальных элементов.

```
SUM_BEF, SUM_AFT=0;
DO I=1 BY 1 TO 30 WHILE(A(I) ¬=0);
    SUM_BEF=SUM_BEF+A(I);
END;
DO I=I+1 BY 1 TO 30;
    SUM_AFT=SUM_AFT+A(I);
END;
```

В качестве начального значения для параметра второго оператора цикла берется последнее значение параметра, оставшееся от первого оператора цикла и увеличенное на единицу.

3.9. Оператор связи с пультом

Оператор связи с пультом аналогичен по своему назначению оператору PAUSE в фортране. С его помощью можно приостановить выполнение программы, выдать на операторский пульт ЭВМ сообщение и после получения ответа продолжить выполнение программы. Например, по оператору

```
DISPLAY('ПОСТАВЬТЕ_КОЛОДУ_#2; ОТВЕТ=ДА')
                                REPLAY(OTVET);
```

на пульт выдается текст

ПОСТАВЬТЕ КОЛОДУ #2; ОТВЕТ=ДА

и выполнение программы приостанавливается; после того как с пульта будет выдан ответ ДА, он в виде значения символьной строки присваивается символьно-строчной переменной OTVET, описанной в программе, и выполнение программы продолжается.

В фортране приведенному оператору соответствует оператор вида

```
PAUSE 'ПОСТАВЬТЕ_КОЛОДУ_#2'
```

Для продолжения выполнения программы на фортране необходимо с пульта дать стандартный ответ.

Форма.

оператор-связи-с-пультом~
~DISPLAY(*выражение-символьное*)
[REPLAY (*переменная-символьная*)]

В качестве символьного выражения в частном случае может использоваться символьная переменная или константа, рассмотренные в гл. 1; длина их не должна превышать 72 символов. Символьное выражение рассматривается в гл. 8.

Если конструкция с REPLAY отсутствует, то печатается сообщение на пульте и выполнение программы продолжается без паузы.

Поскольку оператор DISPLAY сильно замедляет выполнение программы, на его использование в каждой организации обычно накладывают ограничения.

3.10. Оператор останова

Оператор останова имеет вид

STOP;

Он может служить для окончания выполнения программы. Этот оператор в PL/1 не является обязательным: выполнение программы прекращается и после того, как выполнится последний ее оператор (таковым всегда является END-оператор).

3.11. Простейшие операторы ввода-вывода

Здесь описываются простейшие операторы ввода-вывода, которые используются в PL/1 для ввода исходных данных с перфокарт и вывода результатов на печать. Основные возможности операторов ввода-вывода рассматриваются в гл. 6.

Ввод с перфокарт исходных данных осуществляется обычно оператором вида

GET LIST ({*переменная* | *имя-массива*},..);

Как и в фортране, переменная может быть с индексами, а в случае, если задано имя массива, то вводятся значения для всех элементов массива.

Вводимые значения на 80-колонных перфокартах имеют вид чисел или строчных констант без повторителей (см. 1.7.2), и они должны разделяться пробелами (одним или несколькими); для многомерных массивов вводимые значения перфорируются в таком порядке, который соответствует расположению элементов матрицы в памяти ЭВМ по строкам, а не по столбцам, как в фортране.

Распределение вводимых значений по нескольким перфокартам — произвольное; в частности, какое-либо значение может даже быть расположено на двух очередных перфокартах (в самых последних колонках одной и в самых первых колонках следующей перфокарты).

Предположим, что на перфокарте пробита следующая информация:

```
1.1__1.6__1.85__1.91__1.97__1.99
                        __2E—4____5_'1'B_'HET'
```

Рассмотрим оператор

GET LIST(A, B, C, T, H);

Если A является матрицей, состоящей из двух строк и трех столбцов, то ее элементам присвоятся первые 6 констант; причем первые 3 константы являются значениями для первой строки матрицы, а последние 3—для второй строки. Переменная B получает значение 2E—4, C становится равной 5, а строчные переменные T и H получают значения '1'B и 'НЕТ'.

Величины A, B и C могут иметь любые арифметические атрибуты (FLOAT, FIXED, точность), а T и H—любую длину. Атрибуты заданных для ввода констант будут при вводе приводиться к атрибутам переменных по тем же правилам, что и в операторе присваивания (см. 3.1).

Наряду с пробелами для отделения значений на перфокартах могут использоваться и запятые; две подряд идущие запятые означают, что пропускается присваивание (ввод) очередной переменной (элементу данных).

Вывод результатов на печать осуществляется обычно оператором вида

PUT LIST({*выражение* | *имя-массива*},..) [SKIP];

Выражения в списке выводимых величин могут быть и арифметическими и строчными; в частном случае они являются переменными или константами. Строки печатаются во внутреннем виде, то есть без окаймляющих апострофов и с заменой двойных внутренних апострофов на одинарные. Количество знаков в печатаемом значении зависит от его точности или длины. Если задается имя массива, то печатаются значения всех его элементов (для многомерных массивов—по строкам).

Печать производится на 120-позиционном бланке в 5 столбцов при условии, что каждое печатаемое значение умещается в 23 позиции; в противном случае количество значений в печатаемой строчке уменьшается.

Ключевое слово SKIP задает печать с новой строчки АЦПУ, оно может быть помещено как после слова PUT, так и после списка выводимых значений. Если слово (опция) SKIP отсутствует, то выводимые по оператору значения будут, в отличие от фортрана, располагаться при печати сразу вслед за значениями, отпечатанными ранее по предыдущему оператору PUT (например, в следующих столбцах на той же строчке печати). Например, по оператору

PUT SKIP LIST('МАКСИМУМ_ИЗ_', A, B, '=', M);

с новой строчки АЦПУ будет напечатана заданная символьная стро-

ка, значения величин A и B, знак равенства и в пятом столбце — найденное ранее максимальное значение из A и B.

Замечание. Операторы с аналогичными возможностями имеются и в фортране 77 (вместо метки формата в них задается звездочка); в списке величин, как и в операторе PL/1, могут присутствовать и выражения.

Упражнения

3.1. Имеются переменные E и T соответственно с атрибутами FLOAT(6) и FIXED(4) и текущими значениями 1.23456E2 и 7890. Определить результаты выполнения следующих операторов присваивания:

а) E = T; б) T = E; в) E = 0.5 — 1.3I; г) T = 0.5 — 1.3I;

3.2. Следующий оператор фортрана:

GO TO(140, 170, 190, 200), N

записать на PL/1.

3.3. Используя оператор GO TO и меточную переменную с индексом, вычислить $z = 0.3y^2 - b$, где y вычисляется по одной из приведенных ниже формул в зависимости от значения переменной k, задающей номер формулы:

1: $y = x + 0.5$; 2: $y = x^3 - 10^{-4}$; 3: $y = bx^2$; 4: $y = \frac{1}{b}x$

3.4. Следующий фрагмент на фортране записать на PL/1:

```
   IF(X .LT. Y) GO TO 10
      T = X
   GO TO 20
10    T = Y
20 V = T*T
```

3.5. Найти максимум из X, Y и Z.

3.6. Найти произведение ста первых натуральных чисел.

3.7. Найти сумму всех элементов вектора a_i ($i = 1, 2, ..., n$).

3.8. Найти сумму всех элементов матрицы b_{ij} ($i = 1, 2, ..., n$; $j = 1, 2, ..., m$).

3.9. Найти произведения отрицательных и положительных элементов вектора c_k ($k = 1, 2, ..., n$).

3.10÷3.12. Для упражнений 3.5, 3.7, 3.8 написать необходимые операторы ввода и вывода.

3.13. Печатать элементы массива Q_m ($m = 1, 2, ..., N$) до тех пор, пока не встретится отрицательный элемент.

3.14. Вводить и печатать введенные значения до тех пор, пока не встретится нулевое значение.

3.15. Извлечь квадратный корень из величины x по итерационной формуле Ньютона

$$y_{n+1}=\frac{1}{2}\left(y_n+\frac{x}{y_n}\right)$$

с точностью $\varepsilon=10^{-8}$. В качестве начального приближения взять

$$\left[y_0=\frac{1}{2}+\frac{[1]}{[2]}x.\right]$$

ГЛАВА 4

ОПИСАНИЕ И АТРИБУТЫ

4.1. Оператор описания

Вместо нескольких операторов спецификации в фортране (операторы описания типа, DIMENSION, COMMON, EQUIVALENCE и т. п.), с помощью которых задаются характеристики используемых в программе (подпрограмме) переменных, массивов и функций, в PL/1 имеется один оператор описания, начинающийся с ключевого слова DECLARE. Например:

```
DECLARE (A, I) FIXED, (P, K, SUM) FLOAT,
        D COMPLEX FLOAT;
```

Как видим, в одном операторе описания можно характеризовать переменные разных типов. В фортране пришлось бы написать несколько операторов:

```
INTEGER A,I
REAL P, K, SUM
COMPLEX D
```

Атрибуты PL/1 задаются вслед за описываемыми именами; список переменных с одинаковыми атрибутами заключается в круглые скобки. Атрибуты точности, длины и размерности, имеющие вид пары круглых скобок с заключенными в них списками констант, переменных или выражений, также могут фигурировать в операторе описания наравне с другими атрибутами, имеющими вид ключевых слов. Например:

```
DECLARE (T, NUM) FIXED(8),
        PROD(20, 20) FLOAT(10), YES BIT(1),
        FAMILE(100) CHARACTER(20),
        E COMPLEX FLOAT(7);
```

Атрибуты (20, 20) и (100), как и в описании типа фортрана, специфицируют PROD и FAMILE как массивы.

Если атрибуты точности отсутствуют при описании арифметических переменных, то предполагается, что они имеют следующие значения:

для FIXED — (5), для FLOAT — (6).

Напомним, что максимальная точность, которая может быть задана, равна:

для FIXED — (15), для FLOAT — (16).

Для строчных переменных атрибут длины всегда должен задаваться явно.

В PL/1, как и в фортране, имеется возможность не описывать переменные, а приписывать им необходимые атрибуты, воспользовавшись правилами умолчания. Но в PL/1 эти правила более сложны, и мы пока (до гл. 7) не будем ими пользоваться.

Операторы описания PL/1, в отличие от соответствующих операторов фортрана, можно располагать в любых местах программы (или подпрограммы), но «выполнение» этих операторов производится всегда в самом начале программы (подпрограммы), до выполнения других операторов. Можно, таким образом, считать, что все операторы описаний поднимаются в ходе трансляции в самое начало подпрограммы.

Вынесение за скобки атрибутов, общих для нескольких переменных, можно осуществлять неоднократно, то есть допускается описание следующего вида:

```
DECLARE (((B, PROD) (20), C(10)) FLOAT(6),
         VOL FLOAT(16)) COMPLEX;
```

Комплексные (COMPLEX) переменные B, PROD и C будут, кроме того, соответственно иметь следующие атрибуты:

(20) FLOAT (6), (20) FLOAT (6), (10) FLOAT (6).

Атрибут размерности должен всегда располагаться первым слева в последовательности атрибутов; он может находиться или сразу вслед за описываемым именем массива или вслед за круглой скобкой, закрывающей список имен массивов (снабженных, может быть, и другими атрибутами). Атрибуты точности и длины должны находиться всегда сразу вслед за соответствующим ключевым словом, то есть оператор

```
DECLARE (((B, PROD) (20), C(10)) (6),
         VOL(16)) FLOAT COMPLEX;
```

был бы ошибочным (ср. с примером выше).

Строение оператора DECLARE можно выразить в следующей форме:

оператор-описания —

~ DECLARE { *имя атрибуты* | ({*имя* [*атрибуты*]} , . .) *атрибуты* | ({({*имя* [*атрибуты*]} , . .) *атрибуты*} , . .) *атрибуты* | <и т. п.> } , . . ;

Замечание. В отличие от фортрана, где одна и та же переменная может быть специфицирована разными операторами описаний (например, в REAL и DIMENSION), в PL/1 каждое имя может встречаться в операторах описания только один раз, где указываются сразу все необходимые атрибуты.

4.2. Основные атрибуты

Напомним здесь те атрибуты, которые были введены ранее в этой книге:

арифметические: FIXED, FLOAT, COMPLEX, (*точность*);
строчные: CHARACTER, BIT, (*длина*);
меточные: LABEL, (*контрольные-метки*);
дополнительные: (*размерность*).

Точность задается одним или двумя целыми числами; длина и размерность могут иметь вид арифметических выражений.

В PL/1, в отличие от фортрана II и IV, индексы могут принимать не только положительные значения, но и нулевые и отрицательные; поэтому в атрибуте размерности PL/1 характеристика диапазона изменения индекса для каждой из размерностей массива может задаваться и с помощью *граничной пары*. Например:

```
DECLARE A(0:99) FIXED,
        (B, C) (—5:+5, —1:N+1) FLOAT,
        (D FIXED(4), E FLOAT(8))
        (1:M, —1:SQRT(M)) COMPLEX;
```

Левая нижняя граница граничной пары должна быть по своей величине не больше, чем соответствующая правая (верхняя) граница. На использование выражений (и переменных) в граничных парах наложены некоторые ограничения, о чем будет сказано позже (см. 7.1); если выражения имеют не целые значения, они приводятся к целым отбрасыванием дробной части. Пока в конкретных примерах и упражнениях будут использоваться только постоянные границы (задаваемые константами).

Напомним, что атрибут размерности должен стоять самым первым вслед за именем массива или за круглыми скобками, заключаю-

щими список имен массивов, может быть, уже и снабженных другими атрибутами (см. пример выше).

Формы.

атрибут-точности ~ (*число-цел* [, *число-цел*])
атрибут-длины ~ (*выражение-арифм*)
атрибут-контрольных-меток ~ (*метка*, . .)
атрибут-размерности ~ ({[*выраж-арифм*:] *выраж-арифм*},..)

4.3. Начальные значения

Начальные значения для переменных PL/1 задаются при их описании с помощью атрибута начальных значений, который имеет следующий общий вид:

INITIAL (начальные-значения)

Например:

```
DECLARE RAZ FIXED(3) INITIAL(1),
        ALPHA(1:20) FLOAT(6) INITIAL((20)0),
        BETA(—100 : 0, —1 : +1)
                        COMPLEX FLOAT INITIAL((303)1+1I);
```

Как видим, механизм присваивания начальных значений в PL/1 такой же, как и в фортране. Сравните, например:

```
INTEGER RAZ /1/
REAL ALPHA(1:20) /20 * 0/
```

Напомним, однако, что многомерные массивы PL/1 располагаются в памяти и инициализируются «по строкам», а не по столбцам, как в фортране. Кроме того, в отличие от фортрана, в PL/1, как правило, тип константы должен соответствовать типу инициализируемой переменной, то есть, например, нельзя арифметической переменной присваивать строчную константу. Но атрибуты длины, точности и другие арифметические атрибуты констант и переменных могут различаться.

Атрибут INITIAL, если он вынесен за скобки, относится, как и все другие атрибуты, ко всем переменным, заключенным в скобки. Например, в следующем описании:

```
DECLARE (M1, M2, M3) FIXED INITIAL(0),
        (P(15), Q(23)) FLOAT INITIAL((20) 1E—6);
```

атрибуты начальных значений относятся сразу к M1, M2, M3 или к массивам P и Q. При инициализации массива P лишние 5 констант в атрибуте не используются, а для Q последние 3 элемента массива останутся неинициализированными. В фортране инициализация не отдельной переменной, а списка допускается только в операторе

DATA и количество констант должно строго соответствовать количеству инициализируемых элементов. Например, приведенному описанию P и Q в фортране могли соответствовать бы следующие операторы:

```
DIMENSION P(15), Q(23)
DATA P, Q /38 * 1E—6/
```

В PL/1 можно оставить отдельные элементы массива неинициализированными, задав для них в атрибуте на нужном месте звездочку вместо константы, например:

```
DECLARE R(15) FLOAT INITIAL((M) *, (15—M) 1E—6);
```

Первым M элементам массива R не присвоится никаких значений, а последние элементы получат заданное значение. Как видно из этого примера, в качестве повторителей в PL/1, в отличие от фортрана, могут использоваться и выражения; переменные, входящие в эти выражения, должны получить значения заранее из другой подпрограммы (см. внешние переменные в 4.4.) или блока (см. глобальные переменные в 7.1).

До сих пор примеры инициализации приводились только для арифметических переменных, но атрибут INITIAL может быть приписан также строчным и меточным переменным. Например:

```
DECLARE PRIZ BIT(1) INITIAL('0'B),
        NAME CHAR(D) INITIAL((D) '#'),
        SAVE(10) CHAR(4) INITIAL((10)(4)'#');
```

Если инициализируется строчный массив, то нужно быть осторожным, так как в ряде случаев приходится употреблять как повторитель исходных данных, так и повторитель строки. Когда задан один повторитель, то он считается повторителем строки. Поэтому если в атрибуте для массива SAVE написать (10) '####', то (10) будет считаться не повторителем исходных данных, а повторителем строки. Вследствие этого проинициализируются не 10 элементов массива SAVE, а лишь один самый первый его элемент (при этом 36 последних символов строчной константы отбросятся). Правильную инициализацию можно также выполнить задав (10)(1) '####'.

Для фортрана в аналогичном случае (см. выше (10)'####') оператором

```
INTEGER SAVE(10) /10 * '####'/
```

будут проинициализированы все 10 элементов массива.

Примеры инициализации меточных переменных (массивов) были даны в 3.2 при описании оператора, аналогичного вычисляемому оператору GO TO.

В отличие от фортрана, повторитель можно задавать не только перед отдельной константой, но и перед списком констант, заключенным в круглые скобки и также содержащем повторители. Например, по описанию

DECLARE MATRIX(20, 20) FIXED INITIAL((19) (1, (20)0), 1);

элементам матрицы, стоящим на главной диагонали, будет присвоена единица, а остальным — нуль.

Особенность. Существенным отличием начального присваивания в PL/1 является то, что инициализация переменных происходит при каждом вызове какой-либо подпрограммы PL/1, в то время как в фортране присваивание начальных значений происходит лишь однажды перед началом выполнения всей программы. Эта особенность не касается внешних переменных PL/1 (см. 4.4), которые инициализируются лишь один раз в начале выполнения программы.

Формы.

атрибут-нач-значений~

~INITIAL (
({*константа* | *}) <для скаляра>
({[(*выражение*)] {*константа* | *}},..) <для массива↓>
({[(*выраж*)] {*конст*|*| ({[(*выраж*)]
{*константа*|*}},..)}},..)
({[(*выраж*)] {*конст*|*| ({[(*выраж*)]
{*конст*|*| ({[(*выраж*)]
{*конст*|*}},..)}},..)}},..)
<и т. д.>
)

Замечания.

1. Параметры (фиктивные аргументы) подпрограмм не могут быть снабжены атрибутом INITIAL при своем описании; в фортране аналогичное ограничение существует для оператора DATA.

2. Специальные операторы, аналогичные оператору DATA и подпрограмме-спецификации BLOCK DATA, которые соответственно используются в фортране для инициализации обычных и общих переменных, в PL/1 отсутствуют.

4.4. Внешние величины

Величины, которые должны быть известны в нескольких подпрограммах (в PL/1 они называются *процедурами* — см. гл. 5), называются в PL/1 *внешними*, и им в каждой подпрограмме приписывается при описании атрибут EXTERNAL; описатель EXTERNAL в фортране имеет несколько другой смысл.

При изменении внешней переменной в одной подпрограмме ее значение меняется и во всех других подпрограммах. Внешние пере-

менные применяются, как правило, для передачи информации между подпрограммами, заменяя в некоторых случаях аппарат параметров-аргументов при вызове подпрограмм.

По своим свойствам внешние переменные PL/1 соответствуют общим (COMMON) переменным в фортране. Например, операторам фортрана

```
COMMON P, Q, R
REAL P, Q, R(40)
```

соответствует следующий оператор PL/1:

```
DECLARE (P, Q, R(40)) FLOAT EXTERNAL;
```

Заметим, что в отличие от фортрана, атрибут EXTERNAL не может быть использован для совмещения в памяти величин, имеющих разные имена, и поэтому порядок описания внешних величин в PL/1 не имеет никакого значения. Точнее, видимо, было бы считать (в терминах фортрана), что каждая внешняя переменная PL/1 располагается в своем помеченном общем блоке, имя которого в PL/1 не указывается. То есть вышеприведенному оператору PL/1 следовало бы поставить в соответствие следующий оператор фортрана:

```
COMMON /BP/ P, /BQ/ Q, /BR/ R
```

Здесь BP, BQ, BR — условные имена общих блоков.

Присваивание начальных значений внешним переменным PL/1 производится с помощью обычного атрибута INITIAL, например:

```
DECLARE (Q INITIAL(0),
         R(40) INITIAL((40)6.66666E0)) FLOAT EXTERNAL;
```

Инициализация внешних переменных PL/1, как и общих переменных фортрана, происходит один раз перед началом выполнения всей программы (составленной из подпрограмм); инициализация обычных «внутренних» (INTERNAL) переменных в PL/1 осуществляется каждый раз при вызове подпрограммы на выполнение.

Внешние переменные, в отличие от общих переменных фортрана, во всех подпрограммах должны иметь одинаковые атрибуты, включая и атрибут INITIAL; при этом разрешается атрибут INITIAL задавать и лишь в одной из подпрограмм.

Атрибуты EXTERNAL и INTERNAL называются атрибутами *сферы* (*области действия*); INTERNAL принимается по умолчанию.

4.5. Совмещаемые величины

С помощью атрибута DEFINED можно совмещать в памяти одни переменные данной подпрограммы с другими. Назначение этого атрибута то же, что и оператора EQUIVALENCE в фортране.

Например, операторам фортрана

```
INTEGER A, X
REAL B(10), Y(10), Z(10)
EQUIVALENCE (A, X), (B, Y, Z)
```

в PL/1 соответствует описание

```
DECLARE (A, X DEFINED A) FIXED,
        (B, (Y, Z) DEFINED B) (10) FLOAT,
        /* ИЛИ (B(10), (Y(10), Z(10))
                              DEFINED B) FLOAT*/;
```

Одна из совмещаемых переменных (см. A и B) считается *базовой*, другие *совмещаемыми* («определяемыми»). Базовая и совмещаемая переменные могут быть описаны в разных операторах описания. В PL/1, в отличие от фортрана, совмещение переменной всегда указывается в том же операторе DECLARE, где производится ее первое (и единственное) описание. Последовательность описания базовой и совмещаемой переменных в подпрограмме не играет никакой роли, то есть можно при желании написать и так:

```
DECLARE X DEFINED A FIXED;
DECLARE A FIXED;
```

Существенным ограничением по сравнению с фортраном является то, что все атрибуты совмещаемых переменных должны совпадать, за исключением атрибута длины для строчных переменных (но длина совмещаемой переменной никогда не должна превосходить длину базовой переменной) и в некоторых случаях — атрибута размерности.

Арифметическую простую (скалярную) переменную можно совместить со скалярной переменной или с одним из элементов массива; в последнем случае базовая переменная задается с индексами (только в оптимизирующем трансляторе). Например,

```
DECLARE QPRIM FIXED(10) DEFINED Q(1, 1),
        Q(1 : 20, 1 : 30) FIXED(10);
```

В программе к элементу Q(1, 1) можно обращаться и по имени переменной QPRIM, совмещенной с этим элементом.

Совмещение строчных (символьных и битовых) переменных можно осуществлять не только с 1-го символа (бита), но и с любого другого, например:

```
DECLARE L CHAR(40), M CHAR(10) DEFINED L,
        N CHAR(30) DEFINED L POSITION(11);
```

На строку L накладываются строка M (с 1-го символа) и строка N (с 11-го символа). Номер позиции задается целым десятичным числом

без знака. Совмещаемые строки не должны выходить за границу базовой строки.

Совмещать массивы можно двумя способами. В простейшем случае, когда массивы имеют одинаковое строение (одинаковую размерность и граничные пары), устанавливается взаимно однозначное соответствие между элементами совмещаемых массивов (см. выше самый первый пример). В более сложных случаях программист может указать для совмещаемого массива необходимое ему соответствие с элементами базового массива. В этом случае массивы могут быть с разными границами и разных размерностей. Соответствие элементов совмещаемых массивов задается с помощью специальных условных переменных, имеющих имена 1SUB, 2SUB, 3SUB и т. д. (1-й индекс, 2-й индекс, 3-й индекс). Имеются в виду индексы совмещаемого массива, которые пробегают все значения от нижней границы до верхней. Например, в описании

```
DECLARE H(20) FIXED, E(10, 10) FLOAT,
        G(10) FIXED DEFINED H(2 * 1SUB),
        F(0 : 9) FLOAT DEFINED E(1SUB + 1, 1SUB + 1);
```

указано, что элементы массива G соответствуют четным элементам массива H, то есть $G(I) = H(2 * I)$, а элементы вектора F соответствуют элементам матрицы E, расположенным на главной диагонали, то есть

$$F(K) = E(K + 1, K + 1).$$

В следующем примере матрица S является транспонированной по отношению к базовой матрице T:

```
DECLARE T(20, 20) FLOAT(10),
        S(20, 20) FLOAT(10) DEFINED T(2SUB, 1SUB);
```

Зависимость индексов базового массива от индексов совмещаемого массива (*i*SUB) должна быть линейной, но вместо констант могут быть в ряде случаев использованы любые арифметические выражения (приводимые к целочисленным).

Строчные массивы (в порядке, так сказать, исключения) могут быть совмещены со строчными скалярными переменными и наоборот, но без использования атрибута POSITION. Например:

```
DECLARE FAM CHAR(400),
        M(400) CHAR(1) DEFINED FAM,
        N(50) CHAR(4) DEFINED FAM;
```

На символьную строку длиной 400 накладываются символьные массивы, содержащие в качестве своих элементов строки длиной 1 и 4 (массив N наложится только на первые 200 символов строки FAM).

Перечислим некоторые ограничения, связанные с совмещением величин в памяти ЭВМ (другие возможности и ограничения будут описаны позже — см. гл. 7).

1. Базовая переменная не может быть сама с атрибутом DEFINED.

2. Совмещаемая переменная не может иметь атрибутов EXTERNAL и INITIAL: она не может быть параметром процедуры (но может быть аргументом) — см. 5.3.

В заключение заметим, что несмотря на общие функциональные черты оператора EQUIVALENCE и атрибута DEFINED их применение в языках может преследовать и разные цели. В фортране такое средство используется, как правило, для экономии памяти путем совмещения по памяти больших массивов с различными характеристиками. В PL/1 наряду с экономией памяти рассматриваемый атрибут применяется часто для переобозначения величин с целью более наглядной и экономной записи алгоритма.

Упражнения

4.1. Определить атрибуты для переменных, встречающихся в следующем операторе:

```
DECLARE (A FIXED, B FLOAT) (7),
        ((C COMPLEX, D)(5) INITIAL((5)0), E) FLOAT(8);
```

4.2. Массиву B(0:20, —10:10), имеющему атрибуты BIT(80), присвоить по INITIAL следующие значения:

а) (80)'1'B — для 1-й и последней строк, а остальным элементам — (80)'0'B;

б) (40)'10'B — для 1-го и последнего столбцов, а остальным элементам — (40)'01'B;

в) '1'B для побочной диагонали, а остальным элементам — '0'B.

4.3. Найти максимальный элемент матрицы Q_{ij} ($i, j = 1, 2, \ldots, 30$) и его индексы. Дать описание используемых величин и операторы ввода-вывода.

4.4. Упорядочить по неубыванию вектор B_i ($i = 1, 2, \ldots, 40$), элементами которого являются символьные строки длиной 25, не содержащие русских букв. Дать описание используемых величин и операторы ввода-вывода.

ГЛАВА 5

ПРОГРАММА И ПРОЦЕДУРЫ

5.1. Программа

Программа PL/1, как и в фортране, состоит из нескольких (в частности, из одного) связанных между собой модулей, один из которых является главным (MAIN). Эти модули называются в PL/1 *процедурами* (в фортране — подпрограммами). Связи между модулями-процедурами осуществляются с помощью аппарата параметров и аргументов, называемых обычно в фортране формальными (фиктивными) и фактическими аргументами.

Все процедуры PL/1 имеют одинаковую структуру:

процедура~заголовок тело конец

Заголовок главной процедуры имеет вид

имя: PROCEDURE OPTIONS(MAIN);

Имя процедуры является внешним именем и не может содержать более 7 символов (см. 1.2). Заголовки других процедур описываются дальше (см. 5.2).

Тело процедуры состоит из описаний и операторов (в фортране — невыполняемых и выполняемых операторов). Какие-либо ограничения на порядок взаимного расположения описаний и операторов (имеющиеся, например, в фортране) в PL/1 отсутствуют. Но в дальнейшем мы будем придерживаться следующего хорошего правила: располагать, как и в фортране, все описания до выполняемых операторов.

Среди операторов в процедурах PL/1 присутствуют и такие операторы, как RETURN; и STOP; , которые оканчивают выполнение процедуры. Но в PL/1 (как и в фортране 77) эти операторы не являются обязательными, так как окончание выполнения процедуры (а для главной процедуры и всей программы) происходит также в случае, когда выполняется оператор конца процедуры.

Оператор *конца* для процедуры имеет вид

END; или END *имя-процедуры*;

Пример. Написать главную процедуру (главную подпрограмму) для нахождения сумм положительных и отрицательных элементов вектора C_k ($k=0, 1, 2, \ldots, 99$).

```
SUMPN: PROCEDURE OPTIONS(MAIN);
       DECLARE C(0:99) FLOAT INITIAL((100) 1E66),
               (SUM_P, SUM_N) FLOAT INITIAL(0),
               K FIXED;
       GET LIST(C);
       PUT LIST('C', C) SKIP;
       DO K=0 BY 1 TO 99;
          IF C(K) > 0
             THEN SUM_P=SUM_P+C(K);
             ELSE SUM_N=SUM_N+C(K);
       END;
       PUT LIST('SUM_P, ␣SUM_N=',
                                  SUM_P, SUM_N) SKIP;
END SUMPN;
```

Инициализация массива C задана для возможных распечаток массива при отладке программы. Печать строк 'C' и 'SUM_P, ␣SUM_N=' должна помочь разбору печатаемых исходных данных и результатов.

Форма.

главная процедура~

~{*имя*:}... PROCEDURE OPTIONS(MAIN);

{*оператор* | *описание*}... END [*имя-процедуры*];

5.2. Процедуры

Процедуры в PL/1 разделяются на процедуры-подпрограммы и процедуры-функции, которые в дальнейшем часто будем называть просто подпрограммами и функциями (ср. в фортране SUBROUTINE и FUNCTION); в литературе по PL/1 процедуры-подпрограммы обычно называют процедурами.

5.2.1. Процедура-подпрограмма. Заголовок для подпрограммы имеет следующий вид:

заголовок-подпрограммы~

~{*имя*:}... PROCEDURE [(*параметр*,..)];

Например:

SUM: PROCEDURE(X, Y, Z);

В фортране аналогичный заголовок имел бы вид

SUBROUTINE SUM(X, Y, Z)

Список параметров (формальных аргументов), как и в фортране может отсутствовать. Параметры, если они есть, должны быть описаны, как и обычные переменные. Подпрограмма PL/1 может иметь несколько различных имен.

Вход в подпрограмму осуществляется по оператору вызова (см. 3.) в нем задается список аргументов, количество которых должно совпадать с количеством параметров, например:

```
CALL SUM(A, B, C);
```

Выход из подпрограммы и возврат в точку вызова производится по END-оператору, ограничивающему подпрограмму, или с помощью задаваемого в любом месте процедуры оператора

```
RETURN;
```

В том модуле, откуда производится вызов подпрограммы, должно быть дано описание имени процедуры или, как говорят, *имени входа*, которое в нашем случае может иметь, например, следующий вид:

```
DECLARE SUM ENTRY(FLOAT(10), FLOAT(10), FLOAT(10));
```

После ключевого слова ENTRY (вход) перечисляются через запятую атрибуты, соответствующие атрибутам параметров в описываемой подпрограмме. В приведенном описании входа предполагается, что в подпрограмме SUM параметры описаны следующим образом:

```
DECLARE(X, Y, Z) FLOAT(10)
```

Ниже приведены примерные тексты главной подпрограммы и вызываемой из нее подпрограммы SUM, вычисляющей сумму абсолютных значений двух величин X и Y.

```
MASUM: PROCEDURE OPTIONS(MAIN);
       DECLARE(A, B, C) FLOAT(10);
       DECLARE SUM ENTRY(FLOAT(10),
                         FLOAT(10), FLOAT(10));
       . . . . . .
       GET LIST(A, B); /* ВВОД А И В */
       PUT LIST(A, B); /* ПЕЧАТЬ А И В */
       CALL SUM(A, B, C);
       PUT LIST('C=', C); /* ПЕЧАТЬ
                             C=ABS(A)+ABS(B) */
       . . . . . .
END MASUM;
SUM: PROCEDURE(X, Y, Z)
     DECLARE(X, Y, Z) FLOAT(10);
     Z=ABS(X)+ABS(Y);
END SUM;
```

Количество элементов в списке после ключевого слова ENTRY должно соответствовать количеству параметров описываемой процедуры. Атрибуты, перечисляемые в этом списке, должны в точности соответствовать атрибутам, приписанным параметрам в подпрограмме. Атрибуту размерности в описании входа соответствует атрибут вида (*), или (*, *), или (*, *, *) и т. д; количество звездочек указывает на размерность массива.

Пример. Составить подпрограмму для получения вектора, каждый элемент которого равен сумме элементов, расположенных на соответствующей строке матрицы n-го порядка.

```
SUMSTR: PROCEDURE(MATRIX, VECTOR, ORDER);
          DECLARE (MATRIX(*, *), VECTOR(*)) FLOAT(8),
                    ORDER FIXED;
          DECLARE(I, J) FIXED;
   CYCL:  DO I=1 TO ORDER;
              VECTOR(I)=0;
              DO J=1 TO ORDER;
                 VECTOR(I)=VECTOR(I)+MATRIX(I, J);
              END;
    END CYCL;
END SUMSTR;
```

(Если не заботиться о мнемоничности имен, то вместо MATRIX, VECTOR, ORDER можно взять и более короткие идентификаторы, например M, V, N).

Описание входа:

```
DECLARE SUMSTR ENTRY((*, *) FLOAT(8),
                                (*) FLOAT(8), FIXED);
```

5.2.2. Функция. Заголовок для процедуры-функции в PL/1 имеет, в общем, тот же вид, что и для процедуры-подпрограммы, но, кроме того, добавляется описание типа возвращаемого значения, который может иметь, например, следующий вид:

```
RETURNS(FLOAT(10))
```

Сравните следующие заголовки для функции в PL/1 и фортране:

```
FACTOS: PROCEDURE(N) RETURNS(FLOAT);
REAL FUNCTION FACTOS(N)
```

Выполнение процедуры-функции с заданными аргументами активизируется указателем функции (см. 1.6), располагаемом в арифметическом (или строчном) выражении.

Окончание работы процедуры-функции и выход из нее производятся по оператору возврата, где задается выражение, значение

которого должно быть присвоено указателю функции для возвращения в точку вызова. Например:

```
RETURN(F * F);
```

В фортране этот оператор может быть эквивалентен, например, следующим операторам:

```
FACTOS = F * F
RETURN
```

Описание входа для процедуры-функции отличается от описанного выше для процедуры-подпрограммы только тем, что после атрибута ENTRY со списком задается еще и атрибут RETURNS, характеризующий возвращаемое значение. Например:

```
DECLARE FACTOS ENTRY(FIXED) RETURNS(FLOAT);
```

Пример. Написать процедуру-функцию для вычисления квадрата факториала от целого n.

```
FACTOS: PROCEDURE(N) RETURNS(FLOAT);
          DECLARE N FIXED,
                  F FLOAT INITIAL(1);
            DO I=2 TO N BY 1;
               F=F*I;
            END;
            RETURN(F*F);
END FACTOS;
```

Описание входа совпадает с приведенным выше.

То, что в фортране называется оператором-функцией, в PL/1 оформляется в виде обычной процедуры-функции и располагается, как и в фортране, внутри вызывающей процедуры среди описаний (вместе с соответствующим описанием входа). Например, следующая оператор-функция фортрана

```
M(X, Y) = SQRT(X*X+Y*Y)
```

в PL/1 имеет такой вид:

```
M: PROCEDURE(X, Y) RETURNS(FLOAT);
     DECLARE (X, Y) FLOAT;
       RETURN(SQRT(X*X+Y*Y));
END M;
```

с описанием входа:

```
DECLARE M ENTRY(FLOAT, FLOAT) RETURNS(FLOAT);
```

Замечание. Внутренние функции PL/1 могут содержать несколько разнообразных операторов (в фортране — только один

оператор присваивания). В PL/1 внутренними могут быть не только функции, но и подпрограммы, что не допускается в фортране

5.2.3. **Оператор входа.** Как и в фортране, процедуры PL/1 могут иметь дополнительные входы, указываемые оператором входа, имеющим, например, следующий вид:

```
DELTA: ENTRY(A, B, C);
```

В фортране аналогичный оператор имеет вид

```
ENTRY DELTA (A, B, C)
```

Для дополнительного входа PL/1, так же как и для основного, должно быть задано его описание, например:

```
DECLARE DELTA ENTRY(FIXED  FIXED, FLOAT);
```

Если обращение к дополнительному входу производится с помощью указателя функции, то в оператор ENTRY и в описание входа необходимо добавить описатель RETURNS для характеристики возвращаемого значения.

Оператор входа, как и в фортране, не может располагаться в условном операторе и в операторе цикла

При входе в процедуру по дополнительному входу в выполняемых операторах не должны встречаться параметры, принадлежащие основному или другим дополнительным входам и не определенные в данном входе. В некоторых версиях фортрана в данном случае для неопределенных параметров брались значения, оставшиеся от предыдущих обращений по основному или другим дополнительным входам.

Замечания.

1. В отличие от фортрана, в PL/1 допускается процедура-функция без параметров (как и в фортране 77).

2. Не следует путать оператор RETURN («возврат») и ключевое слово (атрибут) RETURNS («возвращает») в заголовке процедуры функции и в описании входа. Оператор ENTRY содержит список параметров и служит для указания на точку дополнительного входа в процедуру, а атрибут ENTRY характеризует параметры и содержит список атрибутов для параметров, а не их имена

Формы.

процедура ~ {*имя*:}... PROCEDURE [(*параметр*,..)]
[RETURNS (*атрибут-функции*)];
{*оператор* |*описание*}... END [*имя-процедуры*];

оператор-входа ~ {*имя*:}... ENTRY [(*параметр*,..)]
[RETURNS (*атрибут-функции*)];

оператор-возврата ~ RETURN [(*выражение* <для функции>)];

атрибуты-входа ~ ENTRY[(*атрибуты-параметра*,..)]
[RETURNS (*атрибуты-функции*)]

5.3. Параметры и аргументы

5.3.1. Соответствие параметров и аргументов. Параметры, перечисляемые в заголовке процедуры PL/1, как и в фортране, могут являться в теле этой процедуры именами скалярной переменной, массива, процедуры, а также меточной величины. Аргументы, которые задаются при обращении к процедуре, являются выражениями и их частными случаями — переменными, константами, а также именами массивов, процедур и меток (меточных переменных).

Параметры и аргументы, как и в фортране, должны соответствовать друг другу по своему виду и организации: имени процедуры соответствует процедура, имени массива — массив, имени простой переменной — выражение или меточная величина (метка, переменная). Не допускается случай, когда параметру — имени массива — соответствует элемент массива, что возможно в фортране.

В операторе описания при описании параметра-массива в качестве граничных пар могут быть заданы лишь константы или звездочки, но не переменные или выражения (что допускается в ряде других случаев); это же справедливо и для длины строчного параметра. Когда граничные пары параметров или их длины — константы, то эти же константы должны быть заданы и при описании аргументов-массивов (в фортране такого требования нет). Если в качестве граничных пар (длины) для параметра заданы звездочки, то при описании аргументов-массивов их границы (длина) могут быть заданы и выражениями. К подпрограмме суммирования строк матрицы (см. пример в 5.2.1) можно обратиться, например, оператором

```
CALL SUMSTR(A, D, N);
```

Аргументы-массивы A и D этого оператора могут быть описаны в вызывающей процедуре, например, следующим образом:

```
DECLARE (A(N, N), D(N)) FLOAT(8);
```

Переменная N должна быть описана заранее во внешнем блоке или использованы управляемые массивы, о чем подробнее будет рассказано позже; пока следует пользоваться лишь массивами с постоянными границами. В операторе описания имени входа (см., например, 5.2.1) соответственно задаются константы или звездочки.

Параметры могут быть и арифметического, и строчного, и меточного типов, которым, конечно, должны соответствовать типы аргументов. Арифметические атрибуты, а также атрибуты точности и длины (для строчных величин) могут различаться для аргументов и параметров (в фортране типы и длины аргументов и параметров должны совпадать). При вызове процедуры аргументы приводятся к атрибутам параметров, точнее к тем атрибутам, которые указаны в описании входа данной процедуры. Впрочем, задание

аргументов с отличающимися от параметров атрибутами не всегда допустимо (см. ниже передачу именем или значением). Аргументы A и N обращения к подпрограмме SUMSTR могут быть заданы (или описаны) с любой точностью и с плавающей или фиксированной точкой; аргумент D обязательно должен иметь атрибут FLOAT(8) (объяснение этому см. ниже в 5.3.2).

Если аргументами являются метки или меточные переменные, то их значения могут быть использованы для выхода из подпрограммы в вызывающую процедуру. Однако такой возврат усложняет логику взаимосвязи подпрограмм, и прибегать к нему не рекомендуется.

Пример.

```
OR: PROCEDURE OPTIONS(MAIN);
     DECLARE YES ENTRY(FIXED,LABEL,LABEL);
      . . . . . .
      CALL YES(I, M1, M2);
      . . . . . .
   M1:
      . . . . . .
   M2:
      . . . . . .
END OR;
YES: PROCEDURE(K, MY, MN);
     DECLARE K FIXED, (MY, MN) LABEL;
      . . . . . .
      IF K < 0 THEN GO TO MY;
               ELSE GO TO MN;
      . . . . . .
END YES;
```

Возврат из подпрограммы YES происходит на метку M1 или M2 главной процедуры.

Аналогичная возможность имеется и в фортране ЕС. где она реализуется с помощью меток вида &10, &15 и операторов возврата вида RETURN1, RETURN2 и т. п.

В качестве аргументов в вызываемую процедуру, как и в фортране, можно передать и имена других процедур, в том числе и некоторых встроенных (см. в 9.1 математические функции)

Пример.

Функция:

```
SUMFUN: PROCEDURE(F1, F2, X) RETURNS(FLOAT);
          DECLARE (F1, F2) ENTRY(FLOAT) RETURNS(FLOAT),
                  X FLOAT;
          RETURN(F1(X)+F2(X));
END SUMFUN;
```

Описание входов в вызывающей программе:

```
DECLARE SUMFUN ENTRY(ENTRY(FLOAT) RETURNS(FLOAT),
                     ENTRY(FLOAT) RETURNS(FLOAT),
                     FLOAT) RETURNS(FLOAT),
        FUNUSER ENTRY(FLOAT) RETURNS(FLOAT);
```

Обращение к функции:

```
PUT LIST('SUMFUN', SUMFUN(SIN, FUNUSER, Z));
```

На печать будет выдано значение выражения

SIN(Z)+FUNUSER(Z)

Здесь FUNUSER — функция, составленная программистом.

Заранее перечислять где-либо имена процедур, которые задаются в качестве аргументов в PL/1, не требуется (в фортране имена таких процедур должны быть перечислены в операторе EXTERNAL).

Атрибуты, характеризующие параметры, относящиеся к именам процедуры, имеют следующие формы:

ENTRY [(*список-атрибутов-параметров*)]
RETURNS (*атрибуты-возвращаемого-значения*)

Последний атрибут (RETURNS) задается, если характеризуется функция.

Замечания.

1. Параметры не могут объявляться внешними (EXTERNAL или совмещаемыми (по DEFINED) величинами, но они могут быть базовыми переменными.

2. Инициализация параметров атрибутом INITIAL не допускается

3. Параметры, в свою очередь, могут быть аргументами при обращении к другой процедуре (это обычно не допускается в фортране).

5.3.2. **Передача аргументов.** Как и в фортране, различаются два способа передачи аргументов в вызываемую процедуру: *именем* или *значением* (*по имени* или *по значению*).

Именем в PL/1 передается аргумент, который имеет вид идентификатора (имени функции, массива, скаляра) или переменной с индексами и при условии, что атрибуты аргумента и соответствующего параметра совпадают (тип, точность, длина); какое-либо специальное указание для передачи именем в PL/1 отсутствует. Напомним, что передача именем должна выполняться обязательно для тех аргументов, которые служат для получения результатов работы процедуры, то есть эти аргументы соответствуют выходным параметрам процедуры (см. параметр VECTOR в подпрограмме SUMSTR).

Передача значением происходит в том случае, когда не выполняются условия для передачи именем, то есть: аргумент является выражением или константой, или атрибуты аргумента не совпадают с атрибутами соответствующего параметра; кроме того, для принудительной передачи значением аргумента, являющегося идентификатором, его заключают в круглые скобки. Передача значением переменной используется программистом, если ему нужно обезопасить себя от изменения значения этой переменной в подпрограмме, составленной, например, кем-либо другим. При передаче аргумента значением в памяти машины размещаются дубли аргументов (фиктивные аргументы), поэтому, например, передача значением для больших массивов мало эффективна (трёбует много памяти).

При каждом обращении к подпрограмме SUMSTR (см. 5.2.1) массив-аргумент, соответствующий массиву VECTOR, должен иметь атрибуты FLOAT(8), в противном случае заданный массив-аргумент не получит никаких значений (они будут присвоены его дубликату). Для экономии памяти имеет смысл, чтобы и аргумент, соответствующий параметру MATRIX, имел атрибуты FLOAT(8). Атрибуты для аргумента, соответствующего параметру ORDER, могут и отличаться, то есть, например, могут быть другой точности.

Например, описания аргументов и обращение к подпрограмме могут иметь следующий вид:

```
DECLARE M(1:20,1:20) FLOAT(8),
        V(1:20) FLOAT(8),
        N FIXED(3);
CALL SUMSTR(M, V, N);
```

или

```
CALL SUMSTR(M, V, 20);
```

Замечание. В фортране передача именем или значением определяется не видом аргумента, как в PL/1 (или его соотношением с параметром), а видом параметра: именем всегда передаются те аргументы, чьи параметры являются именем массива или процедуры, а также если имя параметра-переменной заключено в разрезы (/ и /); в других случаях (например, для переменной) аргумент в разных трансляторах может передаваться именем или значением в зависимости от его вида.

Формы.

аргумент — {*выражение* | *имя-переменной* | *имя-процедуры* | *метка* | *переменная-меточная* | (*аргумент*)}

параметр — *имя*

5.4. Рекурсия

В PL/1, в отличие от фортрана, из некоторой процедуры допускается обращение к той же самой процедуре. То есть, вызванная ранее из какого-либо блока процедура, не закончив своей работы, вызывается снова и в новом экземпляре начинает выполняться с самого начала; таких повторных (вложенных) обращений может быть несколько. Заканчивается выполнение вызванных процедур в обратном порядке: сначала оканчивается выполнение последнего вызванного экземпляра процедуры, потом предпоследнего и в самом конце — первого.

Те процедуры, которые будут вызываться рекурсивно, снабжаются в заголовке атрибутом RECURSIVE (при описании соответствующего имени входа он не должен указываться).

Классическим примером использования рекурсивной процедуры-функции является процедура вычисления факториала, например:

```
FACTORI: PROCEDURE(N) RECURSIVE RETURNS(FLOAT);
         DECLARE N FIXED;
         DECLARE FACTORI ENTRY(FIXED)
                                    RETURNS(FLOAT);
           IF N=0 ! N=1
              THEN RETURN(1);
              ELSE RETURN(N*FACTORI(N-1));
END FACTORI;
```

Рассмотрим процесс выполнения данной процедуры на примере. Пусть указатель функции, с помощью которого произошел вызов процедуры, имеет вид FACTORI(3). В ходе выполнения процедуры (до ее завершения) из оператора RETURN произойдет повторное обращение к процедуре по указателю FACTORI(2). Опять же до завершения выполнения процедуры происходит еще один (третий) вызов процедуры для указателя FACTORI(1). В этом случае процедура выполняется до конца и происходит возврат из процедуры по оператору RETURN(1). Возвращенное значение умножается на N, равное в данном экземпляре процедуры двойке, и значение 2*1 будет возвращено в первый экземпляр процедуры FACTORI. Там оно умножится на 3 и по оператору RETURN (3*2*1) указатель FACTORI(3) получит значение 6.

Рекурсивные процедуры, в частности, могут быть с успехом применены в различных областях вычислительной математики. Например, с их помощью удобно вычислять кратные интегралы, для чего в фортране обычно приходится использовать несколько различных подпрограмм из пакета типовых подпрограмм.

5.5. Задание

После того как программа составлена, ее текст для трансляции и выполнения на ЭВМ должен быть включен в *задание*. В простейшем случае, когда программа состоит из одной готовой подпрограммы, задание может иметь следующий вид:

```
//имя-задания  JOB . . . . . .
//             EXEC PL1LFCG
//PL1L.SYSIN DD *
  текст-исходной-программы
//GO.SYSIN    DD *
  исходные-данные-для-GET
//
```

Имя задания не может содержать более 8 буквенно-цифровых символов. Сочетание //, характеризующее директивы задания, должно располагаться в первых двух позициях. Пробелы обязательно должны окаймлять ключевые слова JOB, EXEC, DD; в других местах директив (управляющих карт) пробелы, как правило, не допускаются. Операторы исходной программы могут начинаться со 2-й позиции и кончаться на 72-й. Исходные данные программы могут располагаться в любых позициях на бланке.

В случае когда программа состоит из нескольких модулей-процедур, задание имеет, например, следующий вид:

```
//имя-задания     JOB . . . . . .
//SH1             EXEC PL1LFC
//PL1L.SYSIN      DD *
  текст-модуля-1
//SH2             EXEC PL1LFC
//PL1L.SYSIN      DD *
  текст-модуля-2
//SH3             EXEC PL1LFCLG
//PL1L.SYSIN      DD   *
  текст-модуля-3
//GO.SYSIN        DD   *
  исходные-данные
//
```

На первом и втором шагах (SH1 и SH2) задания производится трансляция двух модулей, а на третьем (SH3) шаге — трансляция (C — Compile) 3-го модуля, объединение (L — Link) всех модулей в программу и выполнение (G — Go) полученной программы с заданными исходными данными. Головной модуль может транслироваться на любом шаге.

Приведенные задания можно назвать условными, так как при разработке реальных программ их тексты в исходном (или в уже оттранслированном) виде обычно записываются в библиотеки, находящиеся на магнитных дисках, откуда и вызываются для трансляции и объединения. Когда исходные тексты модулей хранятся на диске, в приведенном задании вместо звездочек должны быть заданы характеристики соответствующих наборов данных (и имена разделов — если используются библиотечные наборы данных); тексты модулей из задания, таким образом, убираются.

Как и в фортране ЕС, трансляцию (и выполнение) всех процедур можно осуществить в одном шаге, поместив их все после директивы с именем PL1L.SYSIN. Но в PL/1 требуется, чтобы отдельные процедуры были разделены управляющим оператором *PROCESS, располагаемым с 1-й колонки и в простейшем случае имеющим вид:

*PROCESS;

Таким образом, вместо текста исходной программы в первом задании могут фигурировать и несколько процедур в следующем виде:

текст-модуля-1
*PROCESS;
текст-модуля-2
*PROCESS;
текст-модуля-3

После слова *PROCESS могут быть заданы и необходимые опции транслятора в обычном виде. Нужно заметить, что при использовании оператора *PROCESS, в случае грубой («терминальной») ошибки в одной из процедур, остальные подпрограммы транслироваться не будут (в отличие от случая, когда процедуры транслируются в разных шагах).

Замечание. Главная процедура PL/1, в отличие от фортрана, может иметь параметр (только один). Этот параметр должен быть описан со следующими атрибутами (атрибут VARYING объясняется в 8.2.1):

CHARACTER(100) VARYING

Аргумент для этого параметра, имеющий вид символьной строки, задается в директиве EXEC, например, следующим образом:

имя EXEC PL1LFCLG,PARM.GO='ВАРИАНТ␣1'

В теле главной процедуры переданный аргумент может быть использован произвольным образом, например в условном операторе:

```
PRIM: PROCEDURE(P) OPTIONS(MAIN);
        DECLARE P CHAR(100) VARYING;
        . . . . . .
        IF P='ВАРИАНТ␣1' THEN CALL PP1;
```

Упражнения

5.1. Написать программу для решения упражнения 4.3.

5.2. Составить программу для транспонирования матрицы (20×20).

5.3. Оформить решение упр. 3.9 в виде подпрограммы; дать описание входа.

5.4. Оформить решение упр. 3.8 в виде функции; дать описание входа.

5.5. Оформить в виде функции вычисление значения $\cos x$ с заданной точностью ε, по следующему ряду:

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots$$

Вычисления производить, пока член ряда превосходит величину ε, задаваемую при каждом обращении к функции.

ГЛАВА 6

ВВОД-ВЫВОД

В PL/1 имеется два способа ввода-вывода: потоком (STREAM) и записями (RECORD). Первый способ осуществляется операторами GET и PUT, второй — операторами READ и WRITE. При первом способе данные на внешних носителях («в потоке») могут иметь только символьное представление, и поэтому, например, во время ввода (вывода) арифметических данных происходит их преобразование во внутреннее машинное представление (или наоборот). При втором способе вид вводимой и выводимой информации (расчленной на отдельные порции — *записи*) соответствует внутреннему машинному представлению, и обмен происходит без какого-либо преобразования. Первый способ ввода-вывода применяется в основном при вводе исходных данных с перфокарт и для печати результатов работы программы, второй способ — при обмене промежуточными результатами с магнитными носителями (дисками, лентами). Первый способ, в частности, соответствует форматному вводу-выводу фортрана, второй способ — бесформатному и прямому.

Сначала подробно рассматривается ввод-вывод потоком, а затем излагаются основы ввода-вывода записями (см. 6.5).

6.1. Операторы ввода-вывода потоком

Операторы ввода-вывода потоком имеют следующий общий вид:

$\left\{\begin{matrix}\text{GET}\\\text{PUT}\end{matrix}\right\}$ (*спецификация данных*) (*опции*);

Ломаные скобки указывают на то, что в операторе должна присутствовать по крайней мере одна из двух конструкций.

Имеется 3 вида управления вводом-выводом:

а) управление списком (LIST);

б) управление данными (DATA);

в) управление редактированием (EDIT).

В соответствии с этим общая форма для операторов ввода-вывода получает следующий вид:

$$\left\{\begin{matrix}\text{GET}\\ \text{PUT}\end{matrix}\right\}\left[\begin{matrix}\text{LIST }(\textit{элементы-данных})\\ \text{DATA }[(\textit{элементы-данных})]\\ \text{EDIT }(\textit{элементы-данных})\ (\textit{список-форматов})\end{matrix}\right](\textit{опции});$$

Ниже даются общие сведения, относящиеся ко всем трем видам управления вводом-выводом. Примеры пока будут даваться для управления списком (LIST), кратко рассмотренного ранее (3.11). Элементами вводимых данных, задаваемыми в операторе ввода (GET), являются переменные или имена массивов. Для оператора вывода (PUT) элементами являются выражения (в частном случае — константы, переменные, указатели функций) или имена массивов. Отметим, что в списке выводимых величин в фортране допускаются только переменные и массивы, но не выражения (и не константы).

И при вводе, и при выводе (для DATA — только для вывода) элементом данных может быть DO-элемент, аналогичный циклическому элементу в фортране. Этот элемент позволяет, например, выводить (или вводить) компоненты массивов в произвольном порядке и в любом количестве.

Пример.

```
GET LIST(A, B, C);
PUT LIST(((A(I, J), B(I, J) DO I=1 TO N),
                                C(J) DO J=1 TO N));
```

Первый оператор вводит значения матриц A и B по строкам, а также значения вектора C. Второй оператор выводит на печать элементы матриц A и B парами и по столбцам и в конце вывода каждого из столбцов печатает соответствующий элемент вектора C.

В фортране аналогичный список выводимых элементов (без внешних скобок) выгляделбы так:

```
((A(I,J),B(I,J),I=1,N),C(J),J=1,N)
```

Основной компонентой DO-элемента PL/1 является заголовок цикла, то есть DO-оператор с параметром цикла (без точки с запятой); в нем может присутствовать и конструкция с WHILE и список цикла (см. 3.8).

Форматы, используемые в PL/1, перечисляются в 6.2.

Помимо опции SKIP, рассмотренной в 3.11, имеются еще и другие опции; все они рассматриваются ниже (*e* обозначает арифметическое выражение, приводимое перед выполнением опции к целому значению). Опции можно располагать и до ключевых слов LIST, DATA, EDIT; нужно помнить, что задаваемые ими действия всегда выполняются до начала ввода-вывода указанных данных. Например,

в обоих операторах

PUT LIST(P, Q, R) SKIP; и PUT SKIP LIST (P, Q, R);

переход на новую строчку печати осуществляется до начала печати.

Опции для ввода.

COPY — контрольная печать вводимых данных. Каждое введенное значение печатается в том виде, в котором оно отперфорировано на перфокарте. Каждое отдельное значение печатается с новой строчки, поэтому для больших массивов использовать эту опцию следует осторожно.

SKIP[(*e*)] — пропуск *e* перфокарт, считая и ту, с которой производится ввод. Следующие данные будут вводиться, начиная с 1-й позиции подведенной перфокарты. Если (*e*) не задано, то предполагается, что $e=1$ и осуществляется подво к началу следующей перфокарты. Если $e \leqslant 0$, то полагается $e=1$.

Опции для вывода.

PAGE — подвод первой строчки следующей страницы; перед началом печати в программе эта опция приводит к пропуску целой страницы. Стандартная страница содержит 60 строчек и располагается в первых строчках 72-строчного бланка печати (между двумя просечками на бумаге).

SKIP[(*e*)] — пропуск *e* строчек на печати, считая и текущую. Следующие результаты будут печататься, начиная с 1-й колонки печати. Если (*e*) не задано, то полагается $e=1$ и осуществляется подвод на начало следующей строчки. Если $e \leqslant 0$, то печать будет происходить на текущей строчке, начиная с ее начала, что можно использовать для наложения выводимых по данному оператору символов, на символы, напечатанные предыдущим оператором (или операторами). Если *e* больше количества оставшихся строчек на странице, то вне зависимости от величины *e* просто устанавливается новая страница.

LINE (*e*) — подводится строчка с номером *e*, считая от начала страницы. При $e \leqslant 0$ полагается $e=1$; если *e* меньше или равно текущему номеру строчки на странице, то предварительно устанавливается новая страница. Опция может использоваться и в совокупности с PAGE, например:

PAGE LINE(5)

В соответствии с приведенной ранее общей синтаксической формой в операторах ввода-вывода допускается отсутствие ключевых слов LIST, DATA, EDIT со списками вводимых (выводимых) данных и наличие только одной или нескольких опций. Например, по операторам

GET SKIP(5);
PUT PAGE LINE(5);

пропустится 5 перфокарт на вводе, а на печатающем устройстве подведетс S-я строчка на новой странице.

Замечания.

1. Как уже отмечалось ранее (см. 3.11), если не задана опция SKIP, то в отличие от фортрана ввод-вывод будет производиться на текущей линии (перфокарте или строчке печати), а не на следующей.

2. Возможности операторов ввода-вывода фортрана EC, задаваемые ключевыми словами END и ERR, реализуются в PL/1 с помощью ситуаций ENDFILE и ERROR, которые описываются в 7.3.

Оператор фортрана

READ(5, 20), A, B, C, END=30, ERR=40

в PL/1 можно реализовать, например, следующим образом:

```
ON ENDFILE GO TO L30;
ON ERROR GO TO L40;
 GET EDIT(A, B, C) (R(L20)) SKIP;
```

Операторы ON для некоторых GET или PUT могут быть и опущены в том случае, когда последовательно выполняемые операторы ON содержат одни и те же операторы.

В PL/1 вместо операторов перехода в ON-операторах могут быть использованы и другие операторы (см. 7.3.2).

6.2. Ввод-вывод, управляемый данными

Рассматриваемый тип ввода-вывода во многом аналогичен вводу-выводу с использованием оператора NAMELIST в фортране.

Ввод-вывод, управляемый данными, позволяет вводить и выводить значения, снабженные именами соответствующих переменных (элементы массивов снабжаются и индексами, имеющими вид констант). Например, на перфокарте вводимая информация может иметь следующий вид:

X=2.5, Y(3,8)=1E—4, Y(3,9)=—44E5;

или

B='1'B, C='YES', D=1.5—2.4I;

Запятая может быть заменена или окружена пробелами; пробелы могут окаймлять знак равенства (присваивания) и другие знаки. Все числовые константы (в том числе и комплексные) не должны содержать пробелов; в строчных константах нельзя использовать повторители. В конце информации, относящейся к одному оператору ввода, должна быть точка с запятой. Атрибуты задаваемых констант могут и не совпадать с атрибутами соответствующих переменных; в процессе ввода производятся необходимые преобразования вводимых данных.

На печать значения переменных выдаются в таком же виде, но без запятых; печать, как и для LIST, производится в 5 столбцов (в фортране ЕС по NAMELIST — в 8 столбцов); в конце выводимой порции информации печатается точка с запятой.

Оператор ввода имеет следующий вид:

GET DATA [(*список-имен-переменных*)];

В случае когда список имен (скаляров или массивов) в операторе присутствует, то, как и в фортране, имя каждой переменной, имеющееся на вводимых перфокартах, сравнивается с именами в списке и, если такого имени в списке не оказывается, фиксируется ошибка (ситуации) типа NAME и присваивание данной переменной не производится. Если список имен в операторе отсутствует, то контроль имен при вводе не производится.

Ситуация NAME возникает также в том случае, когда имя, заданное на внешнем носителе, не известно в том месте процедуры, где выполняется соответствующий оператор ввода. Данный тип ввода удобно, например, применять, когда среди вводимых значений элементов матрицы много равных нулю.

Оператор вывода имеет вид

PUT DATA [(*список-переменных*)];

Если список в операторе присутствует, то на печать выдаются значения указанных величин с их именами (для массивов — список их элементов с индексами). Если список переменных отсутствует, то на печать выдаются значения всех переменных, которые описаны (явно или неявно) в той подпрограмме, где находится оператор вывода. Точнее, учитывая блочное строение подпрограммы (см. 7.1.2), печатаются значения всех переменных, существующих в том месте подпрограммы, где находится оператор вывода.

Ограничение. В списке имен, указанных в операторе, или среди имен на перфокартах не могут присутствовать имена, являющиеся параметрами подпрограммы.

Пример.

```
DECLARE FUN CHAR(7), SW BIT(4), NUM(0:2) FIXED,
        R FLOAT INITIAL(1E—10), SUM FLOAT;
 GET DATA;
 GET DATA(SUM, SW);
 PUT DATA(SW, FUN, SUM, NUM);
```

Пусть во входном потоке находятся следующие данные:

FUN = 'SIN'␣NUM(0) = 15␣NUM(2) = —444␣NUM(1) = 197;
SW = '1'B, R = 1E—6, SUM = 43.873563;

Первый оператор GET вводит данные для FUN и NUM; второй — для SW, R и SUM; при вводе значения для R возникнет

ситуация NAME, напечатается предупреждающая диагностика и соответствующее поле данных будет пропущено. По оператору PUT на печать выдается информация следующего вида (с точностью до количества пробелов между отдельными значениями — печать происходит в 5 столбцов):

```
SW='1000'B␣...FUN='SIN␣␣␣␣'␣...
                                    SUM=␣4.38735E+01␣...
NUM(0)=␣␣␣␣␣␣␣15␣...NUM(1)=␣␣␣␣␣␣197␣
                                    ...NUM(2)=␣␣␣␣␣−444;
```

Замечание. Основными особенностями ввода-вывода, управляемого данными, является то, что в PL/1 при выводе имена выводимых переменных задаются непосредственно в операторе вывода, а в фортране — в операторе NAMELIST, на который дается ссылка из оператора вывода. Кроме того, в PL/1 каждое значение и при вводе и при выводе снабжается своим именем (для массивов — с индексами), а в фортране к одному имени массива приписывается несколько значений (при вводе в фортране используются и повторители). Строчки (перфокарты) начала и конца данных, помечаемые в фортране символом &, в PL/1 отсутствуют.

6.3. Ввод-вывод, управляемый редактированием

6.3.1. Операторы. Сначала рассмотрим те операторы редактируемого ввода-вывода в PL/1, которые аналогичны операторам фортрана и имеют следующий вид:

{GET / PUT} (EDIT (*список-данных*) (R (*метка*))) (*опции*);

Метка принадлежит оператору формата, где, как и в фортране, дается список форматов:

оператор-формата ~ *метка*: FORMAT (*список-форматов*);

Например, операторы PL/1

```
    PUT EDIT(A, B(I), C) (R (L10)) SKIP;
L10: FORMAT(E(10,4), F(9,3), F(5));
```

задают вывод, аналогичный выводу по следующим операторам фортрана II:

```
    PRINT 10, A, B(I), C
10 FORMAT(E10.4, F9.3, I5)
```

В фортране IV EC оператор будет иметь вид

```
WRITE(6,10) A, B(I), C
```

В отличие от операторов фортрана форматы в PL/1 можно располагать не только в операторе FORMAT, но и в самом операторе

ввода-вывода, например:

```
PUT EDIT(A, B(I), C) (E(10,4), F(9,3), F(5)) SKIP;
```

Опция SKIP в примере добавлена для того, чтобы вывод происходил с новой строчки, как это всегда производится в операторах вывода фортрана. В рассматриваемых операторах могут быть использованы все опции, изложенные ранее. Сравнение форматов PL/1 и фортрана дается ниже.

6.3.2. Форматы данных. Форматы PL/1, их вид, назначение, принцип действия во многом аналогичны форматам фортрана. Ниже при описании форматов применяются те же обозначения, что и в фортране:

w — количество считываемых или печатаемых символов, ширина (width) обрабатываемого поля;

d — количество цифр (digit) в дробной части числа или в его мантиссе (на перфокарте, бланке печати).

В качестве w и d в PL/1 могут задаваться выражения (вычисляемые в момент использования соответствующего формата), в то время как в фортране они должны быть числами.

Формат с плавающей точкой. Формат для данных с плавающей точкой имеет следующий вид:

формат-плавающий ~ E (w, d [, s])

⟨то есть E (w, d) или E (w, d, s)⟩

Если пока не рассматривать необязательный параметр s, используемый только при выводе, то можно сказать, что данный формат аналогичен формату в фортране, имеющему вид E $w.d$.

Отметим следующие особенности.

При вводе пробелы внутри вводимого числа не допускаются; число плавающего, фиксированного или целого типа может располагаться в любом месте поля, определяемого по w (в фортране — только у правого края); окаймляющие пробелы на числовое значение не влияют. Как и в фортране, буква E перед порядком может не указываться, но знак порядка в этом случае является обязательным. Кроме того, напомним, что если точка имеется во вводимом числе, то d во внимание не принимается.

Примеры.

В потоке	Формат	Значения
−1234567E890	E(10,6)	−1.234567E8
1234.567E890	E(10,6)	+1234.567E8
␣␣1234567E8␣	E(12,4)	+123.4567E8
␣−1234567+8␣␣	E(11,4)	−123.4567E8
␣1234567E8␣␣	E(7,4)	+12.3456

Если бы из потока (с перфокарт) поступили значения вида 1234.567E␣8 или ␣—␣1234567E8, то это привело бы к ошибкам (типа CONVERSION), так как внутри числовых констант находились бы пробелы. В фортране входные данные и соответствующие форматы для третьего и пятого примеров привели бы к ошибкам (в одном случае порядок берется 80, в другом — вводится целое значение, а не вещественное, что недопустимо).

При в ы в о д е, в отличие от фортрана, старшая цифра ненулевой мантиссы всегда отлична от нуля. Если s не задано, то точка располагается справа от первой (старшей) цифры. Параметр s ($s \leqslant 16$) задает общее количество печатаемых цифр, то есть, $s-d$ определяет необходимое количество цифр в целой части печатаемого значения (если s отсутствует, то полагается $s=d+1$).

Как и в фортране, производится округление выводимого значения или добавление нулей в случае, когда заданное количество выводимых цифр (s или $d+1$) меньше или больше количества цифр в печатаемом значении. Как и в фортране, вообще говоря, должно выполняться соотношение $w \geqslant d+7$ (для случая когда $s=d+1$) или $w \geqslant s+6$ (если параметр s задан в формате); в противном случае может возникнуть ошибка.

З а м е ч а н и я.

1. Знак порядка в PL/1 выводится всегда.

2. Формат D в PL/1 не используется: для ввода-вывода значений любой точности используется формат E.

П р и м е р ы.

Значение	Формат	В поток
—1.234567E3	E(14,5)	␣␣—1.23457E+03
+1.234567E3	E(14,8)	1.23456700E+03
—1.234567E3	E(14,8)	ошибка
+1.234567E3	E(14,3,5)	␣␣␣␣12.346E+03

В PL/1, если не принять должные меры, ошибка приводит к прекращению выполнения программы; в фортране подобная ошибка ($w < d+7$) приводит к печати w звездочек.

Формат с фиксированной точкой. Формат для данных с фиксированной точкой имеет следующий вид:

формат-фиксированный ~ F (w [, d [, t]])

⟨то есть F(w) или F(w, d) или F(w,d,t)⟩

Если пока не принимать во внимание необязательный параметр t, то можно сказать, что данный формат заменяет форматы фортрана, имеющие вид Iw и F$w.d$. Формат F(w) обычно служит для работы с целыми значениями, а F(w,d) — со значениями, содержащими дробную часть (целые в PL/1 — частный случай величин с фиксированной точкой).

При вводе вводимое из потока значение может иметь вид целого или фиксированного числа (без порядка), окруженного пробелами. Как и в фортране, если вводимое число содержит точку, то d игнорируется. Пробелы внутри вводимого числа не допускаются. Параметр t задает степень десятки (10^t), на которую нужно умножить введенное значение (ср. с форматом P в фортране); при отсутствии параметра полагается $t=0$.

Примеры.

В потоке	Формат	Значение
—1234␣␣7890	F(6)	—1234
␣␣1234567800	F(7)	+12345
—1234567890	F(8,4)	—123.4567
␣1234.5678␣␣	F(12)	+1234.5678
1234.5678	F(8,1)	+1234.567
␣1234567␣␣	F(10,3,—2)	+12.34567

В фортране при использовании соответствующего формата I в первом примере введется 12340, а в четвертом будет зафиксирована ошибка из-за несоответствия типа вводимого значения и формата; для последнего примера формат имел бы следующий вид:

2PE10.5 или 2PF8.3

При выводе, как и в фортране, нужно обеспечить, чтобы все выводимые в поток символы числового значения умещались в w позициях. Для дробных значений должно обеспечиваться условие $w \geqslant d+3$, то есть в общем случае предусматривается вывод точки, знака минус и нуля перед точкой. Для значений, содержащих дробную и целую части, учитывается возможность увеличения количества цифр в выводимом значении из-за его округления (для значений вида 99.99). По фиксированному формату могут выдаваться и величины с плавающей точкой.

Примеры.

Значение	Формат	В поток
+1234.567	F(11,4)	␣␣1234.5670
—00123.4567	F(10,2)	␣␣␣—123.46
—1234.567	F(7)	␣␣—1235
—.1234567	F(5,3)	ошибка(—0.123)
+1234.567	F(10,5,—2)	␣␣12.34567

В фортране формат I7 в третьем примере привел бы к ошибке так как выдается не целое, а вещественное значение; в четвертом примере не выполняется условие $w \geqslant d+3$.

Формат комплексный. Для ввода-вывода комплексных значений в PL/1 применяется комплексный формат, имеющий вид

C(*формат-действит*[,*формат-действит*])

Соответствующего формата в фортране нет. Под действительным форматом понимается формат F или E. Первый формат в скобках относится к действительной части комплексного значения, второй — к мнимой; если в скобках задан один формат, то он относится как к действительной, так и к мнимой частям.

Из входного потока считываются или в выходной поток записываются два числовых значения без буквы I после мнимой части.

Пример.

По оператору

GET EDIT(E, D) (C(E(10,3), F(7,3)), C(F(4,2)));

и для потока, содержащего

␣␣—5.678E—3␣12345␣+8762543

комплексные переменные E и D с атрибутами FLOAT(5) и FIXED(4,2) получат следующие значения:

E = —5.6780E—3 + 12.345E0I; D = 08.76 + 25.43I;

В фортране соответствующий список форматов в операторе FORMAT имел бы следующий вид:

(E10.3, F7.3, F4.2, F4.2)

Общее замечание. В PL/1 не требуется соответствия атрибутов вводимых или выводимых арифметических значений типам форматов и атрибутам соответствующих переменных (при вводе) или выражений (при выводе). Например, целое значение можно ввести по плавающему формату и присвоить переменной с фиксированной точкой. В разных версиях фортрана имеются различные ограничения на типы вводимых или выводимых значений и виды форматов. С другой стороны, в PL/1 отсутствует формат, аналогичный формату G в фортране, по которому форма печатаемых значений определяется автоматически по типу и величине значения.

Битовый формат. Битовый формат имеет следующую форму:

формат-битовый ~B[(*w*)] ⟨B(*w*) или B⟩

При вводе *w* указывается в формате обязательно. Введенные из потока *w* символов должны представлять собой последовательность единиц и нулей, может быть, окаймленную пробелами; пробелы внутри битовой строки не допускаются. Если $w \leqslant 0$, то вводится пустая строка. В частном случае последовательность может состоять из одного бита, представляющего истину или ложь; в этом случае можно сказать, что данный формат аналогичен формату L в фортране.

Примеры.

В потоке	Формат	Значение
101101	B(4)	'1011'B
␣␣1011␣	B(7)	'1011'B
␣␣1011␣	B(2)	ошибка
␣␣0␣␣␣	B(6)	'0'B

В фортране последнему примеру соответствуют данные вида ␣␣F␣␣␣ (или ␣FALSE) и формат L6.

При выводе битовых (логических) значений они располагаются у левого края заданного поля из w символов, а не у правого, как в фортране. Если длина (l) выводимой строки больше w, то неумещающиеся правые биты отсекаются; если длина меньше w, то выводимая строка дополняется справа пробелами. Для вывода можно и не указывать w, тогда предполагается, что оно равно длине выводимого битового значения. Для $w \leqslant 0$ выдается пустая строка.

Примеры.

Значение	Формат	В поток
'1011'B	B(6)	1011␣␣
'1011'B	B(3)	101
'1011'B	B	1011
'1'B	B(3)	1␣␣

В фортране при выводе значения .TRUE. по соответствующему формату L3 получим на печати ␣␣T.

Замечание. Битовый формат в совокупности с функцией (псевдопеременной) UNSPEC (см. 9.1.3) может быть использован для ввода-вывода значений переменных во внутренней кодовой форме. В этом случае можно считать, что он аналогичен по своему назначению формату Z в фортране. Например, для следующего описания переменной T

```
DECLARE T CHAR(4) INITIAL('ИВАН');
```

по оператору

```
PUT EDIT(UNSPEC(T)) (B);
```

получим на печати в двоичной кодировке (см. 9.6.)

11001011110000101100000111001000

В фортране для того же значения T по операторам

```
 PRINT 4, T
4 FORMAT(1X, Z8)
```

получим на печати в шестнадцатиричной системе

CBC2C1C8

Символьный формат. Формат для ввода-вывода символьных данных имеет следующий вид:

формат-символьный ~A[(*w*)] ⟨A(*w*) или A⟩

При в в о д е вводимая информация представляет собой произвольную последовательность символов, имеющихся на внешних устройствах ЭВМ («символы ЭВМ»). Параметр w задается при вводе обязательно. При $w \leqslant 0$ вводится пустая строка.

П р и м е р ы.

В потоке	Формат	Значение
␣␣ABCDEFGH␣	A(8)	'␣␣ABCDEF'
1+X*Y−5	A(5)	'1+X*Y'

В ы в о д строчно-символьной информации (как и битовой) происходит во «внутреннем» виде, то есть без окаймляющих кавычек и без удвоения внутренних кавычек. Если длина выводимого строчного значения не совпадает с w, то происходит отсечение символов или добавление пробелов с п р а в а. Если w не задано, то оно принимается соответствующим длине выводимого строчного значения.

П р и м е р ы.

Вид значения	Формат	В поток
'␣ABCDEF␣'	A(4)	␣ABC
'1234'	A(7)	1234␣␣␣
'ABCD␣'	A	ABCD␣

Имеется прямая аналогия рассматриваемого формата с форматом A в фортране. Отличием является лишь то, что в PL/1, в случае, если w больше длины переменной, при вводе отсекаются символы или при выводе добавляются пробелы с п р а в а, а не слева, как в фортране. Например, в фортране для второго примера и формата A7 вывод будет иметь следующий вид: ␣␣␣1234.

З а м е ч а н и я.

1. В PL/1 символьные константы при выводе задаются среди выводимых данных (см. 6.1), а не среди форматов, как в фортране. Например, оператору PL/1

```
PUT EDIT('ПЕРЕМЕННАЯ␣P␣=␣', P) (A, F(5,2));
```

в фортране соответствуют следующие операторы:

```
      PRINT 10, P
10    FORMAT(1X, 15HПЕРЕМЕННАЯ␣P␣=␣, F5.2)
```

или

```
      WRITE(6,10) P
10    FORMAT(1X, 'ПЕРЕМЕННАЯ␣P␣=␣', F5.2)
```

2. Формирование форматов типа H, применяемое в ряде версий фортрана, в PL/1 не допускается.

3. В PL/1 символьные строки при вводе могут присваиваться, как правило, только символьным переменным; в фортране такого ограничения нет.

6.3.3. Управляющие форматы.

Формат пропуска позиций. Этот формат, аналогичный по своему назначению формату wX в фортране, имеет следующий вид:

формат-пропуска ~X(w)

Формат установки позиции. Для того чтобы подвести нужную колонку на перфокарте при вводе или позицию на строчке при печати, используется формат, имеющий следующий вид:

формат-позиции ~ COLUMN(w)

Если номер задаваемой в формате позиции меньше текущего номера, то перед установкой требуемой позиции происходит подвод следующей перфокарты или строчки печати. Если w задает несуществующий номер позиции, например $w > 80$ для перфокарты или $w > 120$ для печати, то полагается $w = 1$. Аналогичным форматом фортране является формат T (исключая его действие в упомянутых особых случаях).

Формат прогона линий. Под линией подразумевается перфокарта или строчка печати. Формат имеет следующий вид:

формат-прогона-линий ~ SKIP[(w)]

По своему виду и назначению формат совпадает с опцией SKIP. В фортране для перехода на новую перфокарту или строчку печати применяется косая черта (/).

Формат страницы. Формат имеет вид

формат-страницы ~ PAGE

Действие этого формата совпадает с аналогичной опцией (см. 6.1). Заметим здесь, что часто для экономии бумаги на АЦПУ устанавливают такую управляющую перфоленту, что вместо подвода нового бланка печати происходит всего лишь прогон нескольких строчек (6—8).

Формат подвода линии. Формат имеет вид, совпадающий с соответствующей опцией, и осуществляет те же действия:

формат-подвода-линии ~ LINE(w)

Пример.

```
PUT EDIT('ТАБЛИЦА␣СВОДНАЯ', LINE1)
        (PAGE, LINE(5), COLUMN(40),
        A, SKIP(2), X(9), I5);
```

Оператор на новой странице в 5-й строчке и с 40-й колонки печатает заголовок, а затем в 8-й строчке с 10-й колонки печатает значение целой переменной LINE1.

Общие замечания.

1. В PL/1 при выводе потоком отсутствует имеющаяся в фортране возможность управления печатью с помощью первого символа, выводимого по оператору вывода. Все управление печатью в PL/1 осуществляется только опциями или управляющими форматами.

Ниже приводится соответствие управляющих символов фортрана форматам (опциям) PL/1.

Фортран	␣	0	+	1
PL/1	SKIP	SKIP(2)	LINE(0)	PAGE

2. Ввод или вывод по оператору происходит с того места на внешнем носителе, перед которым произошло окончание выполнения предыдущего оператора ввода или вывода; в фортране ввод или вывод для каждого оператора всегда начинается с новой перфокарты или строчки печати. Если программист допустил ошибку и выводимые данные не умещаются в строчку печати, то в PL/1 печать переносится на следующую строчку (в фортране неумещающиеся символы отсекаются или фиксируется ошибка).

3. В PL/1 отсутствует имеющаяся в фортране ЕС возможность ввода текста форматов с перфокарт в символьный массив, изменение в процессе выполнения программы и использование его в качестве последовательности форматов в некотором операторе ввода-вывода. Изменение форматов в ходе выполнения программы PL/1 можно осуществить только изменением значений параметров (w, d, s и т. п.), которые в PL/1 могут иметь вид выражений. Кроме того, для замены форматов в ходе вычислений может служить следующая возможность.

Удаленный формат. Общая форма для удаленного формата, который был упомянут в 6.3.1, имеет следующий вид:

формат-удаленный ~ *R* ({*метка* | *переменная-меточная*})

Метка или значение меточной переменной определяют метку оператора FORMAT, в котором задается необходимый список форматов. Удаленный формат может располагаться среди других форматов в операторах PUT, GET и FORMAT. Он может применяться, например, для изменения форматов в ходе выполнения программы или для использования одного и того же списка форматов как части в различных операторах ввода-вывода.

ример.

```
DECLARE M LABEL(LS, LN);
. . . . . .
 IF G¬=0 THEN M=LS; ELSE M=LN;
 PUT EDIT(P,Q, R, S, T) (E(15,7), R(M), F(6,4), A);
LS: FORMAT(F(5),A(11));
LN: FORMAT(F(8), A(8));
PUT EDIT(U, V, X, Y, Z)^(R(LN), B(1), R(LS));
```

В зависимости от значения переменной G величины Q и R будут выдаваться соответственно в форматах F(5), A(11) или F(8), A(8). Переменные U, V, X, Y, Z печатаются в форматах F(8), A(8), B(1), F(5), A(11).

6.3.4. Повторение форматов. В качестве повторителя формата в PL/1 может быть использовано не только целое число, но и выражение (что не допускается в фортране). Если повторителем является выражение, то оно обязательно заключается в круглые скобки; если повторителем является число, то оно заключается в скобки или отделяется от формата пробелом. Повторитель, как и в фортране, можно ставить не только перед форматом, но и перед списком форматов, заключенным в скобки. Например,

```
FN: FORMAT(2 F(5), (3) F(4,2), (2*N) (E(15,8), 2 E(14,7)));
```

Выражение 2*N является повторителем для списка форматов, следующего далее и заключенного в скобки. Значение выражения вычисляется непосредственно перед использованием его при выполнении операции ввода-вывода, то есть N могло бы быть введено в том же операторе ввода-вывода по одному из предыдущих форматов, например по F(5). Если значение повторителя меньше или равно нулю, то формат (список форматов) игнорируется.

В случае, когда форматов, заданных в списке форматов (с учетом повторителей), не хватает для обеспечения всех вводимых (выводимых) данных, форматы повторяются (может быть, и многократно), начиная с самой внешней открывающей скобки в списке форматов оператора ввода-вывода или в операторе FORMAT. (Напомним, что в фортране форматы в этом случае повторяются, начиная с самой правой открывающей скобки.) Например, оператор

```
FORMAT(2 F(5), 2 (E(15,5), 2 E(16,8)));
```

в PL/1 задает следующую последовательность форматов:

F(5), F(5), E(15,5), E(16,8), E(16,8), E(15,5), E(16,8), E(16,8), F(5), F(5) и т. д.,

в то время как в аналогичном операторе фортрана повторились бы только форматы E, но не F. Ограничений на количество вложенных списков форматов, заключенных в круглые скобки, в PL/1 практически не существует (до 20 вложений). Нужно отметить, что в PL/1 перевода на новую линию (перфокарту или строчку) печати в данном случае не происходит. Поэтому, например, операторы

```
    PUT EDIT(P) (R(LP));
LP: FORMAT(E(15,8));
```

или оператор

```
PUT EDIT(P) (E(15,8));
```

могут быть использованы для вывода всех элементов массива (по 8 значений в одной строчке печати для данного примера).

В PL/1 допускается, кроме того, и оператор следующего вида:

```
PUT EDIT(P) (E(15,8)) (Q) (F(12,4)) (R) (E(20,10));
```

который выводит массивы P, Q, R в указанных форматах. То есть более общая форма для оператора ввода-вывода с редактированием, такова (ср. с 6.3.1):

{GET / PUT}(EDIT{(*список-данных*)(*список-форматов*)}...)(*опции*);

Замечание. Действие, задаваемое управляющим форматом PL/1, выполняется только в том случае, если за ним срабатывает формат данных, находящийся в том же списке; в противном случае он просто пропускается. Например, в операторе

```
GET EDIT(S, T) (A(18), X(10), E(12,4), X(40));
```

при выполнении его в цикле формат X(40) будет игнорироваться, и ввод величин S и T с перфокарт будет производиться начиная со следующих позиций: 1, 29, 41, 69, 1, 29, 41, 69 и т. д. (на перфокарте 80 позиций). Для того чтобы ввод S происходил каждый раз с новой перфокарты, можно предложить несколько разных способов:

```
GET EDIT(S, T) (A(18), X(10), E(12,4)) SKIP;
GET EDIT(S, T) (COLUMN(1), A(18), X(10), E(12,4));
GET EDIT(S, T, #) (A(18), X(10), E(12,4), A(40));
GET EDIT(S, T, @) (A(18), X(10), E(12,4), X(39), A(1));
```

Вспомогательные величины # и @ в данном случае могут иметь любые атрибуты, например CHAR(1); при выводе в аналогичных случаях они должны содержать пробелы или быть пустыми.

Пример.

Для величин P, V, S и C, которые имеют, например, атрибуты FIXED(2), FIXED(8,3), BIT(4), CHARACTER(20), вводить с перфокарт значения, расположенные соответственно в колонках 1÷2, P÷P+10, 40÷43, 60÷79, и печатать V, S, C на одной строчке бланка печати в позициях соответственно 20÷29, 40÷43, 50÷69. Над каждым столбцом печатаемых значений напечатать имя соответствующей переменной. Признаком конца ввода является значение P, равное нулю.

```
PUT EDIT('V', 'S', 'C') (COLUMN(25), A, COL(42), A, COL(55), A);
DO WHILE(P¬=0); /*ПРИ ВХОДЕ В ЦИКЛ P¬=0 ! */
   GET EDIT (P, V, S, C) (COL(1), F(2), COL(P),
                          F(11,3), COL(40), B(4), COL(60), A(20));
   IF P¬=0 THEN
```

```
PUT EDIT(V, S, C) (SKIP, COL(20),
                   F(10, 3), COL(40), B(4), COL(50), A);
END;
```

Значение V на перфокарте может и не иметь десятичной точки.

6.4. Замена носителей

6.4.1. Обмен с файлами. Язык управления заданиями ОС ЕС ЭВМ позволяет легко заменить одни носители вводимой-выводимой информации на другие, путем замены характеристик используемых наборов данных в соответствующих DD-картах. В PL/1 принято, что для операторов GET и PUT характеристики используемых в программе наборов данных задаются в директивах DD соответственно с именами SYSIN и SYSPRINT (в фортране ЕС — FT05F001 и FT06F001), которые обычно имеют следующий вид:

```
//SYSIN DD *
//SYSPRINT DD SYSOUT=A
```

Звездочка указывает на то, что вводимая информация находится на следующих перфокартах задания; SYSOUT=A указывает на то, что вывод будет происходить на печать с помощью системного вывода. Для того чтобы осуществлять обмен с другими носителями, например с наборами, расположенными на магнитных лентах, дисках или перфоленте, нужно вместо * и SYSOUT=A указать характеристики наборов данных и тип устройства ввода-вывода. Диспозиция (DISP) при этом должна быть указана обязательно; кроме того, необходимо учитывать, что эти наборы данных обязательно должны иметь следующие характеристики:

для SYSIN — RECFM = F, BLKSIZE = 80;
для SYSPRINT — RECFM = VA, BLKSIZE = 129.

Чтобы сохранить директивы со стандартными именами (SYSIN и SYSPRINT) для ввода с перфокарт и вывода на печать, можно наборы данных для обмена с другими носителями характеризовать в DD с другими (нестандартными) именами. В этом случае выбранное имя DD задается в операторе GET или PUT с использованием ключевого слова (опции) FILE, например:

```
PUT FILE(KOEF) EDIT(P, Q, R) (E(15, 5)) SKIP;
```

или

```
GET FILE(FAM) LIST(PROF);
```

Здесь KOEF и FAM — имена DD, в которых указаны характеристики наборов, используемых для ввода и вывода. В случае, если имя файла не указано (нет опции FILE), то по умолчанию для GET берется FILE (SYSIN), а для PUT берется FILE(SYSPRINT), чем мы пользовались раньше. Когда файл вывода имеет имя, отличное

от SYSPRINT, то опции (форматы) LINE и PAGE запрещаются, если не принять специальной меры. Такой специальной мерой является описание или открытие файла (см. 6.5.1) с атрибутом PRINT. Этот атрибут указывает на то, что вывод на внешние носители (может быть, и на магнитный диск или магнитную ленту) должен производиться в форме, предназначенной в конце концов для печати. В частности, в PRINT-файле в начале каждой строчки присутствует символ, управляющий расположением этой строчки на бланке печати (см. 6.3.3, общее замечание 1).

Напомним еще раз, что при использовании оператора PUT выводимые значения подвергаются преобразованию в символьную форму (как и в фортране для форматного вывода).

Опцию FILE удобно использовать в том случае, когда при печати все данные, выводимые в разное время по некоторому оператору (или операторам) вывода, необходимо на бланке печати собрать вместе. Например, в результате выполнения следующего фрагмента

```
DO WHILE(X < XKOH);
 . . . . . .
   PUT FILE(F) LIST(A, ...);
    . . . . . .
   PUT FILE(G) EDIT(B, ...) (форматы);
    . . . . . .
   PUT FILE(F) DATA(C, ...);
    . . . . . .
END;
```

выведенные в PRINT-файлы результаты будут расположены на бумаге в виде (при соответствующем расположении DD-карт):

A, C, A, C, ... (файл F)
B, B, ... (файл G)

Аналогично и при вводе можно разделить вводимые исходные данные на ряд независимых порций и, поместив их за соответствующими директивами DD, вводить данные независимо из каждой порции.

Замечание. Аналогичные возможности имеются и в фортране ЕС.

6.4.2. Обмен со строкой. В качестве «носителя» вводимых или выводимых данных в PL/1 может быть использована строчная величина, то есть скалярная символьно-строчная переменная или массив. Так что рассматриваемая ниже возможность, строго говоря, к вводу-выводу не относится, хотя и использует его операторы. Например, по оператору

```
GET STRING(ST) EDIT(P, Q, R) (A(5), A(5), A(6));
```

ввод значений для присваивания их переменным (или массивам

P, Q, R будет происходить из символьной переменной ST (находящейся в оперативной памяти).

Пусть задано описание

```
DECLARE (P, Q, R) CHAR(6) INITIAL('123456'),
   ST CHAR(17) INITIAL('ЧТО?␣ГДЕ?␣КОГДА?␣');
```

Тогда по приведенному оператору переменные P, Q, R получат соответственно значения

ЧТО?␣␣, ГДЕ?␣␣, КОГДА?

Таким образом, оператор GET с опцией STRING может быть удобен при разделении строки на несколько составных частей.

Оператор PUT с той же опцией можно применить для объединения нескольких строк в одну. Например, по оператору

```
PUT STRING(ST) EDIT(P, Q, R) (X(2), 3 A(5));
```

для указанных начальных значений P, Q, R (см. выше) строка ST получит значение

␣␣123451234512345

Конечно, переменные, которые фигурируют в операторах ввода-вывода с опцией STRING, могут иметь и арифметический, и битовый тип и редактироваться с помощью форматов E, F, C, B; при этом производится преобразование значений в символьную форму (или наоборот), как и при обычном вводе-выводе, рассмотренном ранее.

Можно сказать, что опция STRING, как и опция FILE, указывает на источник или приемник данных, передаваемых по операторам ввода-вывода. Строка, задаваемая в опции STRING, как бы имитирует входной или выходной поток, представляющий собой по определению последовательность символов ЭВМ. Операторы ввода-вывода с опцией STRING удобно применять для стандартных преобразований строк, используя спецификацию EDIT.

Замечания.

1. При использовании опции STRING никакие другие опции и управляющие форматы в операторе не допускаются (это не относится к управляющему формату X).

2. Если при вводе из строки или выводе в нее оказывается, что ее длина недостаточна, возникает ошибка (ситуация) типа ERROR и выполнение программы прекращается.

3. Каждый раз при выполнении очередного оператора ввода-вывода «ввод» или «вывод» начинается с начала строки.

4. Строка опции STRING соответствует внутреннему файлу в фортране 77.

6.5. Ввод-вывод записями

Обработка записей в наборах данных может производиться *последовательным* или *прямым* способами.

6.5.1. Последовательный ввод-вывод. Последовательный ввод-вывод записей осуществляется операторами READ и WRITE следующего вида:

оператор-ввода ~
~ READ FILE (*имя-файла*) INTO (*имя-переменной*);
оператор-вывода ~
~ WRITE FILE (*имя-файла*) FROM (*имя-переменной*);

Отметим сразу, что в отличие от операторов фортрана здесь можно задавать для обмена всего лишь одну переменную, что, конечно, является ограничением. Это ограничение обычно можно снять, если при обмене использовать структуры данных (см. 7.2).

В качестве переменной в операторах нельзя использовать параметр, совмещаемую переменную (с DEFINED) и структуру, содержащую строки с изменяемой длиной (см. 8.2.1).

Имя файла (содержащее не более 8 символов) в простейшем случае указывает на имя директивы DD языка управления заданиями, в которой заданы характеристики обрабатываемого набора данных. Пока будем считать понятие файла синонимом набора данных. Примеры операторов ввода и вывода записями:

```
DECLARE MAN(40) CHARACTER(30);
. . . . . .
READ FILE(FAMILY) INTO(MAN);
WRITE FILE(RES) FROM(MAN);
```

Производятся ввод одной записи из файла FAMILY и вывод ее в файл RES; в обоих случаях обмен производится с символьным массивом MAN. Файл RES может быть охарактеризован, например, следующей директивой DD:

```
//GO.RES DD DSN = RASDEL,VOL = SER = CARD1, ...
```

В фортране соответствующие оператор чтения и директива могли бы иметь следующий вид:

```
READ(13) MAN
//GO.FT13F001 DD см. выше
```

В PL/1, как правило (если формат записей набора данных отличен от VS или VBS), количество байт в обмениваемой переменной должно точно совпадать с установленной длиной записи, иначе будет зафиксирована ошибка (типа RECORD).

Для окончания работы с файлом выполняется оператор CLOSE, имеющий вид

оператор-закрытия-файла ~ CLOSE (FILE(*имя-файла*)), ..;

Например,

CLOSE FILE(RES), FILE(FAMILY);

По оператору CLOSE соответствующий набор данных устанавливается на начало, на первую запись (это существенно для наборов, расположенных на магнитных лентах). В фортране аналогичным оператором является оператор REWIND, например

REWIND 13

Операторов, аналогичных BACKSPACE и ENDFILE, в PL/I нет, но имеется ряд других, дополнительных возможностей.

Для того чтобы пропустить несколько записей при поиске нужной, выполняется оператор чтения с опцией IGNORE. Например, по оператору

READ FILE(FAMILY) IGNORE(N+1);

будет пропущена N+1 запись.

Если набор данных расположен на магнитном диске и его записи имеют формат F, то есть возможность изменить последнюю считанную запись с помощью оператора REWRITE:

оператор-перезаписи ~

~ REWRITE FILE (*имя-файла*) FROM (*имя-переменной*);

Например, по оператору

REWRITE FILE(FAMILY) FROM(MAN_OLD);

последняя запись, считанная ранее по READ, будет заменена в файле на значение переменной MAN_OLD. Дополнительно требуется, чтобы имя файла было описано в операторе описания с атрибутами FILE и UPDATE («исправить»). Например:

DECLARE FAMILY FILE UPDATE;

Аналогичное описание может быть дано и для любых других файлов. В качестве атрибутов, характеризующих описываемые файлы, могут быть заданы, например, следующие:

INPUT
OUTPUT
UPDATE

Атрибут INPUT характеризует входной файл, то есть такой, из которого можно производить только чтение. Файл, описанный с OUTPUT (выходной), может быть использован только для записи информации в него. К файлу с UPDATE можно обращаться только операторами READ и REWRITE.

Если при описании файла явно не объявляется характер его обработки, то, как и в фортране, это определяется по виду первого оператора, выполняемого над этим файлом (набором данных): для READ устанавливается INPUT, для WRITE—OUTPUT.

Кроме того, тип файла можно задать и в операторе открытия файла:

оператор-открытия-файла ~ OPEN{FILE(*имя-файла*)[*опции*]},..;

Например:

```
OPEN FILE(FAMILY) UPDATE,
     FILE(RES) OUTPUT;
```

Файл FAMILY открывается как исправляемый, файл RES—как выходной; над файлом FAMILY можно производить только операторы READ и REWRITE, а над файлом RES—операторы WRITE. После закрытия файла оператором CLOSE этот же файл может быть снова открыт с другими атрибутами. Причем это может быть сделано как оператором OPEN, так и неявно операторами READ и WRITE. Разумеется, при этом устанавливаемые характеристики не должны противоречить характеристикам (атрибутам), указанным в операторе DECLARE.

В PL/1 имеется возможность читать набор данных на магнитной ленте с конца, с последней записи. Для этого файл должен быть описан (или открыт) с атрибутом BACKWARDS, например:

```
DECLARE NEW FILE INPUT BACKWARDS;
```

Как и в фортране, имя файла (в фортране—ссылочный номер) может быть задано переменной (в PL/1—символьной), которой в ходе выполнения программы можно присваивать различные символьные значения, задающие необходимые имена директив DD. Упомянутая переменная указывается в операторе OPEN с помощью опции TITLE («название»). Например,

```
DECLARE N CHAR(2), MAN FILE;
  IF  T=0   THEN N='FY';
            ELSE N='FN';
  OPEN FILE(MAN) TITLE(N) UPDATE;
```

В зависимости от значения величины T файлу MAN будет соответствовать DD или с именем FY или FN.

Пример. С помощью вспомогательного файла W переписать в обратном порядке файл F с длиной записей, равной 80.

```
DECLARE (F, W) FILE, FF BIT(1), R(20) FLOAT;
ON ENDFILE(F) FF='0'B;
ON ENDFILE(W) FF='0'B;
```

```
EF='1'B;
READ FILE(F) INTO(R);
DO WHILE(EF); /* ПЕРЕПИСЬ ИЗ F В W*/
 WRITE FILE(W) FROM(R);
 READ FILE(F) INTO(R);
END;
CLOSE FILE(F), FILE(W);
OPEN FILE(W) BACKWARDS INPUT RECORD,
     FILE(F) OUTPUT RECORD;
EF='1'B;
READ FILE(W) INTO(R);
DO WHILE(EF); /* ПЕРЕПИСЬ ИЗ W В F */
 WRITE FILE(F) FROM(R);
 READ FILE(W) INTO(R);
END;
CLOSE FILE(F), FILE(W);
```

В первом случае открытие файлов F и W производится неявно с помощью операторов READ и WRITE (современная методика не рекомендует поступать так); во втором случае открытие файлов осуществляется оператором OPEN. Для обнаружения конца файла используется ситуация ENDFILE; в ON-операторе производится присваивание переменной EF, управляющей выполнением оператора цикла, логического значения **ложь.**

Замечание. Операторы READ и WRITE могут быть использованы для ввода (вывода) данных, находящихся на перфокартах, или для печати на АЦПУ. Но так как ввод-вывод записями производится без преобразования, то упомянутая возможность относится только к символьным данным, которые при вводе-выводе не требуют никакого преобразования из внутреннего вида во внешний (или наоборот). При вводе с перфокарт: формат — F, длина блока — 80; при выводе на АЦПУ: формат — VA, длина блока — 129.

6.5.2. Прямой ввод-вывод. Кроме наборов данных с последовательной организацией, когда обработка записей возможна только по очереди, в порядке их расположения в файле, в PL/1 допускаются наборы данных, обработка записей которых может производиться в любом порядке, или, как говорят, *прямым доступом* (DIRECT). Ниже рассматривается работа с наборами данных, организованными *регионально* и характеризуемыми ключевым словом REGIONAL.

Региональный набор состоит из *регионов.* Для наборов REGIONAL(1) каждый регион представляет собой запись (которая не может быть сблокирована); регионы считаются перенумерованными, начиная с нуля. Такие наборы имеют аналог в фортране ЕС и фортране 77.

Количество регионов в наборе данных определяется автоматически по значению параметра SPACE в директиве DD (в котором задан объем внешней памяти, отведенной под набор), а также по формату и длине записи, задаваемыми там же или в атрибуте ENVIRONMENT («оборудование»). В фортране ЕС количество записей в наборе задается явно при описании набора данных (файла). В PL/1 в том же атрибуте ENVIRONMENT задается и ключевое слово REGIONAL(1), например:

```
ENVIRONMENT(REGIONAL(1) F(60))
```

В директиве DD в параметре DCB должен быть указан подпараметр DSORG=DA (прямая организация набора).

В операторе описания среди прочих необходимых атрибутов должен присутствовать атрибут DIRECT, характеризующий прямой ввод-вывод (атрибут RECORD при этом можно опустить). Например:

```
DECLARE F FILE DIRECT OUTPUT
                ENVIRONMENT(REGIONAL(1) V(90));
```

Создание регионального набора данных начинается с открытия выходного (OUTPUT) файла (по оператору OPEN или неявно). При этом система определяет количество регионов и в первый байт всех регионов заносит 8 битовых единиц, то есть байт получает значение, равное (8)'1'B, что является признаком *пустого* региона. Первоначальное заполнение набора данных производится с помощью оператора WRITE вида

```
WRITE FILE (имя-файла) FROM (переменная)
                        KEYFROM (выражение-симв);
```

Выражение, преобразуемое к символьному виду, задает номер региона (берутся 8 правых цифр-символов). Указанный регион заполняется значением переменной (в частности, массивной или структурной — см. 7.2 и 8.3), указанной после FROM; заполнение регионов может производиться в любом порядке.

После создания набора над его записями в режиме UPDATE могут производиться операции добавления, чтения, исправления и стирания соответственно по операторам вида:

```
WRITE FILE (имя-файла) FROM (переменная)
                        KEYFROM (выражение-симв);
READ FILE (имя-файла) INTO (переменная)
                        KEY (выражение-симв);
REWRITE FILE (имя-файла) FROM (переменная)
                        KEY (выражение-симв);
DELETE FILE (имя-файла) KEY (выражение-симв);
```

Опции KEY и KEYFROM задают номера обрабатываемых регионов. Оператор REWRITE позволяет исправить любой регион

(в том числе и пустой), а не только тот, который был перед этим считан, как при последовательном вводе-выводе. Оператор READ читает и пустые записи. Оператор DELETE метит указанную запись признаком пустой записи.

Пример. Не используя вспомогательный набор данных, записи регионального набора данных расположить в обратном порядке (ср. с примером в п. 6.5.1). Пусть набор содержит 400 регионов длиной по 30 байт. Ниже приводится фрагмент программы, решающий поставленную задачу, и примерный вид директивы DD.

```
DECLARE SET FILE DIRECT
                          ENVIRONMENT(REGIONAL(1) F(30)),
 (A, B) CHAR(30),
              /* ВСП. ПЕРЕМЕННЫЕ ДЛЯ ПЕРЕСТАНОВКИ */
 (NA, NB) CHAR(8), /* ИНДЕКСЫ ДЛЯ ПЕРЕСТАНОВКИ */
 (I, K) FIXED;
  . . . . . .
            OPEN FILE(SET) UPDATE;
  CHANGE: DO I=0 BY 1 TO 199;
                NA=CHAR (I, 8); K=399—I; NB=CHAR(K, 8);
                READ FILE(SET) INTO(A) KEY(NA);
                READ FILE(SET) INTO(B) KEY(NB);
                REWRITE FILE(SET) FROM(B) KEY(NA);
                REWRITE FILE(SET) FROM(A) KEY(NB);
  END CHANGE; /* ЗАПИСЬ(I) <=> ЗАПИСЬ (399 — I) */
                . . . . . .
               CLOSE FILE(SET);
                . . . . . .
                . . . . . .
//GO.SET DD UNIT=5061,SPACE=(30,400),DSN=REGNAB,
//          DISP=KEEP,VOL=SER=MYDISK,
//          DCB=DSORG=DA
```

Здесь SET — имя файла, а REGNAB — имя набора данных, находящегося на диске MYDISK. Переменные NA и NB служат для представления номеров переставляемых регионов в символьном виде. Функция CHAR описывается в 9.1.3; в данном случае она необязательна, так как преобразование в символьную форму будет включено автоматически (см. 8.5).

6.5.3. Обобщения. Прежде чем перейти к изложению некоторых дополнительных возможностей ввода-вывода записями, уточним понятие файла в PL/1 и сформулируем его отличие от набора данных (ранее считалось, что эти понятия тождественны).

Набором данных в PL/1 называют совокупность данных, объединенных общим назначением и располагаемых, как правило, на

внешних носителях; наборы данных PL/1 могут иметь одну из организаций, допустимых в языке. Наборы данных существуют реально (физически), и с ними можно работать и вне программ PL/1 (например, по служебным программам или из программ фортрана).

Файл является не физическим, а логическим понятием. В разное время он может соответствовать различным наборам данных, или один и тот же набор данных может быть отождествлен с различными файлами. Файл существует в программе PL/1 только во время ее выполнения, в промежутке от открытия до закрытия файла; набор данных продолжает существовать и после закрытия файла. Отождествление файла с конкретным набором данных происходит при открытии файла.

Иногда говорят, что набор данных является физическим файлом, а файл — логическим набором данных.

В ряде случаев файлы могут иметь другие характеристики, чем у наборов данных. Например, региональный набор данных может быть привязан к последовательному файлу и обработан с помощью соответствующих операторов без указания номеров регионов (без опций KEY и KEYFROM). В этом случае говорят, что региональный набор обрабатывается с помощью последовательного (CONSEQUTIVE) файла или, точнее, путем последовательного («поочередного»—SEQUENTIAL), а не прямого (DIRECT) доступа. Таким образом, атрибуты файла, отождествляемого при его открытии с каким-нибудь набором, определяют характер обработки набора данных (например, последовательно или по номеру региона, чтение, запись или исправление набора). Последовательные файл и доступ (и связанные с ними атрибуты CONSEQUTIVE и SEQUENTIAL, принимаемые по умолчанию), использовались ранее в 6.5.1.

Рассмотрим подробнее использование последовательного доступа при работе с региональным набором данных.

Если создание регионального набора производится с помощью последовательного доступа, то регионы, номера которых задаются в опции KEYFROM, должны заполняться по возрастанию их номеров (в противном случае возникает ситуация KEY). После создания файла к моменту его закрытия во все его незаполненные регионы занесется признак пустой записи. Последовательное чтение из регионального набора данных производится, начиная с нулевой (или последней — для BACKWARDS) записи. Опция KEY в операторе READ не задается, но может быть задана опция KEYTO (переменная), которая присваивает указанной переменной значение текущего номера региона. Например, по оператору

```
READ FILE(H) INTO(V) KEYTO(NUM);
```

символьная переменная NUM получает значение текущего номера региона при обработке файла H (чтения в переменную V).

Исправление региональных наборов также может производиться используя последовательный файл. Например, по программе, приведенной в п. 6.5.1, можно обработать и региональный набор, не меняя ничего в операторах ввода-вывода; лишь описание файла F должно быть дополнено атрибутом ENVIRONMENT с опцией REGIONAL(1); файл W должен располагаться на магнитной ленте.

Упражнения

6.1. Вводить по GET LIST с каждой четной перфокарты по 2 числа и распечатывать их по PUT LIST в 3-м и 4-м столбцах на бланке; во 2-м столбце печатать номера вводимых перфокарт. Посредине бланка над вторым из печатаемых столбцов дать заголовок

ВИД ПЕРФОКАРТ

Для окончания ввода воспользоваться ситуацией ENDFILE (см. 6.1).

6.2. На перфокартах отперфорированы для ввода по GET DATA некоторое количество целочисленных значений массива A (1:4, 1:20). Требуется ввести эти значения и затем напечатать по PUT DATA весь массив, располагая на каждой строчке бланка печати один столбец матрицы A.

6.3. Напечатать таблицу значений $\sin x$, $\cos x$ и $\operatorname{tg} x$ с точностью 8 знаков после запятой для x, изменяющегося от 0° до 45° с шагом в 1°; таблица должна иметь следующий вид:

10 20 40 60 80

X	SIN X	COS X	TG X

6.4. Составить процедуру для печати графика целочисленной неотрицательной функции $f(x)$, значения которой меняются от 0 до M; аргумент x меняется от a до b с шагом $(b-a)/100$.

Ось абсцисс направить по строчке печати, ось ординат — вверх по бланку. Величины a, b, f, M взять в качестве параметров процедуры.

6.5. Имеются 3 символьные строки с длинами 7, 13, 17. Объединить их в одну строку с длиной 40, отбросив предварительно от каждой строки по 2 последних символа и вставив вместо них по 3 пробела.

6.6. Из последовательного файла переписать записи с длиной 24 байта в региональный файл. Три первых байта считанной записи взять в качестве номера региона и в региональный набор не записывать.

ГЛАВА 7

НОВЫЕ ПОНЯТИЯ

В настоящей главе излагается ряд основных понятий PL/1, отсутствующих в фортране. К таким понятиям прежде всего относятся блоки, структуры и ситуации.

7.1. Блоки

7.1.1. Процедурный и простой блоки. Процедура в PL/1, как и подпрограмма (функция) в фортране, является блоком, в том смысле, что система именования величин в одной процедуре (блоке), осуществляемая с помощью описаний, никак не зависит от имен величин, использованных (описанных) в других процедурах (блоках). То есть величины, имеющие одинаковые имена в разных процедурах (блоках), никак не связаны друг с другом, если не приняты специальные меры, например такие, как использование внешних величин (с атрибутом EXTERNAL в PL/1 или COMMON в фортране).

Блоки в PL/1 имеются двух видов: процедурные и простые. *Процедурный блок* представляет из себя процедуру (подпрограмму или функцию). Блоком является не только обычная внешняя процедур , но и внутренняя (см. 5.2.2).

Простой блок имеет следующее строение:

простой-блок ~ [*метка*:]... BEGIN; *описание*...
оператор... END-*оператор*;

Простые блоки в отличие от процедурных могут быть только внутренними: они располагаются всегда внутри некоторой внешней процедуры и могут находиться также внутри внутренних процедур и в других простых блоках.

Процедурные блоки также могут находиться внутри простых блоков. В виде внутренних блоков (процедурных или простых) оформляются некоторые независимые части алгоритма, используемые только в одном из модулей, и в том случае, когда они слишком малы и

просты для того, чтобы их имело смысл выделить в отдельные внешние процедуры.

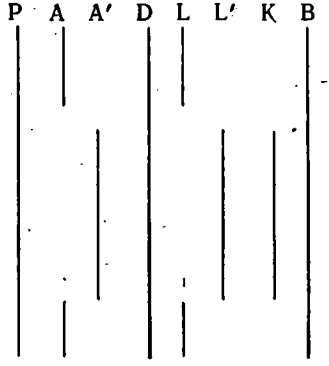
В отличие от процедурного блока, выполнение операторов которого производится по оператору CALL, простой блок начинает выполняться в тот момент, когда закончилось выполнение непосредственно предшествующего оператора (или простого блока), а также в том случае, когда был осуществлен переход на метку блока по оператору GO TO. Отметим, что, в отличие от простых блоков, внутренние процедурные блоки, если они расположены среди операторов процедуры (что допускается в PL/1, хотя и не рекомендуется) последовательно не выполняются, а просто обходятся. Выход из простого блока происходит после выполнения оператора END, ограничивающего блок, или (что обычно не рекомендуется) по оператору GO TO, выполняющему переход на метку, расположенную вне данного блока.

7.1.2. Глобальные величины. Если во внешние блоки (в процедуры) значения величин из других блоков могут передаваться только с помощью аппарата параметров-аргументов или внешних (EXTERNAL) переменных, то для передачи величин во внутренние блоки используются и глобальные имена.

Глобальными называются имена, используемые в некотором блоке, но описанные не в нем, а в окаймляющем блоке.

Рассмотрим следующий пример (справа указаны области действия имен, используемых в примере):

```
P: PROCEDURE;
     DECLARE (A, D) FIXED;
     L: A=1; D=2;
     B: BEGIN;
          DECLARE A FIXED;
             A=D; D=3;
          K: PUT LIST(A, D); /* 2; 3 */
             GO TO L;
               . . . . .
     L: END B;
        PUT LIST(A, D); /* 1; 3 */
END P;
```



Простой блок B входит в процедурный блок P, который может быть или внутренним или внешним. Переменные A, описанные в блоках P и B, являются разными переменными, локализованными каждая в своем блоке, и изменение одной из них на значение другой никак не влияет. Переменная D, описанная в блоке P, и переменная D, встречающаяся в блоке B и не описанная в нем, это одна и та же переменная, и изменение ее значения в одном из блоков повлияет на ее величину в другом. По оператору печати, располо-

женному в блоке P, будет напечатано 1 и 3; тот же оператор печати в блоке B выдает 2 и 3. Переменная D является локальной для блока B и глобальной по отношению к блоку B. Значение глобальной переменной D передается из блока P в блок B (и обратно).

Переход по оператору GO TO будет осуществлен на локальную метку L в блоке B; но если бы метка L в блоке B отсутствовала, то переход произошел бы на глобальную метку в блоке P (метка считается описанной в том блоке, где она метит какой-либо оператор). Вход в блок B на метку K по оператору GO TO K, расположенному в блоке P, невозможен, так как метка K в блоке P неизвестна (она локализована в блоке B). По тем же правилам, что и метки, локализуются и имена процедур (роль GO TO при этом играет оператор CALL).

Использование глобальных имен позволяет ввести в работу массивы с переменными границами или строки с переменной («регулируемой») длиной; в фортране переменные границы массивов возможны только в том случае, когда эти массивы и их границы являются формальными параметрами (в фортране 77 нет такого ограничения).

Пример.

```
Q: PROCEDURE;
     DECLARE (N, M) FIXED;
      GET LIST(N, M);
   D:  BEGIN;
         DECLARE A(-N:N) FIXED,
                 B(0:N, 0:N) FLOAT,
                 C CHAR(2*M+1);
           . . . . . .
   END D;
     . . . . . .
END Q;
```

Вне блока D производится описание величин N и M, а также ввод (или, может быть, какое-либо вычисление их значений). Перед выполнением блока D происходит размещение в памяти описанных в нем величин. Это размещение производится в соответствии со значениями переменных M и N, переданных в качестве глобальных величин из блока Q в блок D. Если бы операторы BEGIN и END блока D отсутствовали, то размещение величин A, B, C («выполнение» оператора описания) происходило бы в самом начале выполнения блока Q, то есть до ввода значений N и M по оператору GET, и привело бы к ошибке.

Если в процедуре Q имеется еще блок (вне блока D), то описанные в нем переменные будут располагаться в памяти на месте, освободившемся после окончания работы блока D (на месте переменных

A, B, C). Таким образом, блочная структура способствует экономии оперативной памяти.

Замечание. Описание имени входа (см. 5.2) для внутренней процедуры должно располагаться в том же блоке, где расположена и сама процедура. Таким образом следующее расположение операторов является ошибочным (квадратные скобки изображают окаймляющие блоки):

```
[[DECLARE      [[PROCEDURE    [[DECLARE
 [CALL          [CALL          [PROCEDURE
  PROCEDURE      DECLARE        CALL
```

В последнем случае ошибка состоит в том, что вызываемая по оператору CALL процедура неизвестна во внешнем блоке.

7.1.3. Атрибуты памяти. Тот способ размещения величин в памяти, который был только что использован для описания массивов с переменными границами, называется *автоматическим*; он характеризуется атрибутом AUTOMATIC, принимаемым по умолчанию. Автоматические величины размещаются в памяти всегда в тот момент, когда начинает выполняться блок, в котором они описаны; память, занятая этими величинами, освобождается по окончании выполнения блока, и значения величин становятся неопределенными.

Другие способы размещения величин в памяти характеризуются атрибутами STATIC, CONTROLLED, BASED.

Статические величины размещаются в памяти один раз до начала выполнения всей программы и не уничтожаются при выходе из блока (их значения сохраняются до следующего входа в блок). Статические величины аналогичны общим величинам фортрана, но существенным отличием является то, что, если не принять специальные меры, статические величины локализуются, то есть известны только в том блоке, где они описаны.

Общие величины фортрана более похожи на внешние величины PL/1 (см. 4.4), которые могут быть известны не в одном только блоке, а в нескольких. Внешним величинам всегда приписывается атрибут STATIC, то есть они размещаются в памяти до выполнения программы. Из этого, в частности, следует, что внешние статические массивы или строки в качестве границ и длин могут иметь только константы (целые десятичные числа).

Для уяснения различий при использовании автоматических, статических и внешних величин рассмотрим следующий пример:

```
PRIM: PROCEDURE;
        DECLARE Z EXTERNAL FLOAT INITIAL(0),
                U STATIC FLOAT INITIAL(0);
      Z=Z+1; U=U+1; PUT LIST(Z, U);
      CALL TWO; PUT LIST(Z, U);
```

```
END PRIM;
TWO: PROCEDURE;
        DECLARE Z EXTERNAL FLOAT INITIAL(0),
                U FLOAT INITIAL(0);
      Z=Z+1; U=U+1; PUT LIST(Z, U);
END TWO;
```

При первом обращении к процедуре PRIM будут напечатаны следующие пары значений переменных Z и U: 1 и 1, 2 и 1, 2 и 1; при втором обращении — 3 и 2, 4 и 1, 4 и 2; при третьем — 5 и 3, 6 и 1, 6 и 3 и т. д. Внешняя переменная Z наращивает свое значение в обеих процедурах PRIM и TWO, и ее значение увеличивается до 6. Статическая переменная U в процедуре PRIM после каждого выхода из процедуры сохраняет свое значение и становится равной 3. Автоматическая переменная U в процедуре TWO после каждого выхода из нее теряет свое значение и при входе каждый раз становится равной нулю в соответствии с атрибутом INITIAL. Атрибут INITIAL для переменной Z может быть убран в одной из процедур, так как инициализация статических и внешних переменных производится лишь один раз перед началом выполнения всей программы.

Размещение в памяти *управляемых* (CONTROLLED) переменных производится по операторам ALLOCATE и FREE, задаваемым пользователем в нужных местах программы. По первому оператору осуществляется размещение управляемых переменных, по второму оператору — их уничтожение. Управляемые переменные удобно использовать для работы с массивами, имеющими переменные границы, или со строками, которые имеют переменную длину; глобальные переменные при этом не используются (ср. с 7.1.2).

Пример.

```
Q: BEGIN;
     DECLARE (M, N) FIXED,
             A(-N:N) FIXED CONTROLLED,
             B(0:N,0:N) FLOAT CONTROLLED,
             C CHAR(2*M+1) CONTROLLED;
   GET LIST(N, M);
   ALLOCATE A, C;
    . . . . . .
   FREE A;
    . . . . . .
   ALLOCATE B;
    . . . . . .
   FREE B, C;
    . . . . . .
END;
```

Размещение массивов A, B и строки C производится не в начале блока, а в моменты, заданные программистом, и уже в то время, когда M и N получили необходимые значения. Освобождение памяти производится по ходу выполнения блока, что к тому же позволяет экономить память, необходимую для выполнения данного блока (например, массив B будет частично размещен системой на памяти, освобожденной от массива A). Если оператор FREE не задан, то освобождение памяти, как и для статических переменных, производится по окончании выполнения всей программы, а не рассматриваемого блока.

В операторе ALLOCATE у имен контролируемых величин можно задать и другие граничные пары (или другую длину), чем в описании; можно задать и атрибут INITIAL, например:

```
ALLOCATE A(-10:10) INITIAL((21)99999), C(L);
```

Размещение будет произведено в соответствии с заданными значениями граничных пар и длины (L должно быть вычислено или введено до выполнения этого оператора).

Формы.

оператор-размещения ~ ALLOCATE
{*имя-переменной*[*размерность*][*длина*][*инициализация*]},..;
оператор-освобождения ~ FREE *имя-переменной*,..;

Замечание. Повторное размещение той же величины по оператору ALLOCATE (до ее уничтожения по оператору FREE) продвигает ее старое значение в магазин («стек») и делает его недоступным до тех пор, пока новое значение этой величины (которая может быть использована в программе обычным образом) не будет уничтожено по FREE. Подробно этот вопрос здесь не рассматривается.

Расположение *базированной переменной* в памяти определяется ее базой — переменной типа *указатель*. Имя связанного с базированной переменной указателя задается при описании переменной, например:

```
DECLARE U FLOAT BASED(P),
        V(10) CHAR(6) BASED(Q);
```

Здесь P и Q — указатели; они могут быть описаны и явно с помощью атрибута POINTER, например:

```
DECLARE (P,Q) POINTER;
```

Базированные переменные не могут иметь атрибутов INITIAL, VARYING и EXTERNAL, а также переменных граничных пар и переменных длин для строчных величин.

Размещение базированных переменных может производиться различными способами, одним из которых является использование

ператора ALLOCATE, например:

```
ALLOCATE U, V;
```

(В отличие от управляемых переменных (см. выше), базированные переменные при размещении не могут снабжаться новыми атрибутами.) После размещения базированной переменной ее указатель получает значение, равное начальному адресу основной (оперативной) памяти, где находится эта переменная. В нашем случае указатели P и Q получат значения начальных адресов величин U и V в памяти.

Значения указателей можно сравнивать между собой по операциям = или ¬=, а также присваивать другим указателям.

Другим способом размещения базированной переменной является присваивание ее указателю некоторого значения. Часто при таком присваивании используется встроенная функция ADDR, которая определяет адрес расположения в памяти заданного аргумента. Например, по оператору

```
Q = ADDR(X);
```

производится определение начального адреса размещенной уже величины X, присваивание его указателю Q и вместе с последним — размещение массива V в памяти, начиная с заданного адреса (принадлежащего X).

Использование базированных переменных позволяет совмещать в памяти переменные различных типов. Например, для описания

```
DECLARE A(5, 10) BIT(48);
```

по оператору

```
Q = ADDR(A(4, 1));
```

массив V разместится в памяти так, что будет наложен на 4-ю строку автоматически размещенной ранее битовой матрицы A. Используя эту строку матрицы, над массивом V можно производить битовые операции.

Уничтожаются базированные переменные по оператору FREE, описанному ранее.

К базированным переменным прибегают при составлении программы из модулей, написанных на разных языках (см. Приложение).

7.1.4. Неявное описание и правило умолчания. Помимо явного описания величин, производимого по оператору DECLARE, в PL/1 допускается и неявное описание (которое не имеет отношения к неявному описанию по оператору IMPLICIT в фортране). При неявном описании принадлежность имени к какому-либо виду величин PL/1 устанавливается транслятором по контексту, в котором встречается не описанное явно имя. Например, если неописанное имя встретилось

после ключевого слова CALL, то оно считается именем процедуры-подпрограммы. Если имя встретилось в выражении и за ним сразу следует открывающаяся круглая скобка, то такое имя считается именем внешней процедуры-функции (со списком аргументов), а не именем массива (со списком индексов). Имя, встречающееся в операторах в том месте, где должно находиться имя файла, считается именем файла. Нужно, однако, отметить, что имя, расположенное после ключевого слова GO TO, неявно, как метка или меточная переменная, не описывается и считается ошибкой. Напомним, что метка считается описанной явно в том блоке, где она встретилась в качестве префикса какого-либо оператора; то же справедливо и для имени блока (приписанного к BEGIN). Имя, встречающееся в выражении или в левой части оператора присваивания, считается скалярной переменной. Скалярными переменными, причем, описанными я в н о, всегда считаются параметры процедуры (и те, для которых явное описание отсутствует). Неявно описанные имена процедур считаются внешними и статическими (EXTERNAL и STATIC), имена переменных — внутренними и автоматическими (INTERNAL и AUTOMATIC), файлов — внешними.

Другие атрибуты величинам (переменным и функциям), описанным неявно, приписываются по правилам умолчания, аналогичным тем, которые приняты в фортране. Если первой буквой неявно описанного имени является одна из букв I, J, K, L, M, N, то величина получает атрибуты

```
REAL FIXED BINARY(15, 0);
```

в противном случае величине приписываются атрибуты

```
REAL FLOAT DECIMAL(6);
```

Двоичные (BINARY) данные описываются в 8.1.2.

Если имя величины присутствует в операторе DECLARE, но для него не указаны н и к а к и е атрибуты, то в этом случае атрибуты такой величине приписываются по сформированному только что правилу. Если при явном описании указаны не все, а лишь некоторые атрибуты, то недостающие выбираются из REAL, DECIMAL, FLOAT, а не из COMPLEX, BINARY, FIXED.

Если при явном описании атрибут точности не задан, то в ЕС ЭВМ он выбирается из следующей таблицы, в которой приведены и максимальные значения, допустимые для атрибута точности.

	по умолчанию	максимум (h)	
FIXED DECIMAL	(5,0)	(15,q)	$-128 \leqslant q \leqslant 127$
FIXED BINARY	(15,0)	(31,q)	
FLOAT DECIMAL	(6)	(16)	
FLOAT BINARY	(21)	(53)	

Например, описание

```
DECLARE MOVE, CORE(15), IR FIXED, B FLOAT,
        OTR COMPLEX, KOR DECIMAL,
        DIG ENTRY(FIXED) RETURNS(BINARY);
```

подразумевает следующие атрибуты для описанных явно переменных:

```
MOVE—REAL FIXED BINARY(15),
CORE —(15) REAL FLOAT DECIMAL(6),
IR   —REAL FIXED DECIMAL(5),
B    —REAL FLOAT DECIMAL(6),
OTR  —COMPLEX FLOAT DECIMAL(6),
KOR  —REAL DECIMAL FLOAT (6),
DIG  —ENTRY(REAL FIXED DECIMAL(5))
                    RETURNS(REAL BINARY FLOAT(21)
```

В последнем случае описывается имя процедуры-функции с одним параметром.

Напомним, что описание входа осуществляется двумя атрибутами (ENTRY и RETURNS), каждый из которых, как оказывается, может отсутствовать. В атрибуте ENTRY характеризуются необходимые атрибуты для аргументов (параметров), а в атрибуте RETURNS указываются атрибуты для значения функции.

Атрибут ENTRY при описании может быть опущен только в том случае, если при всех обращениях к процедуре из данного блока атрибуты всех заданных аргументов в точности совпадают с атрибутами соответствующих параметров (описанных внутри вызываемой процедуры); в противном случае будет происходить ошибочная передача тех аргументов, атрибуты которых не удовлетворяют указанному требованию. Например, если опустить описание функции DIG, то аргументы при всех обращениях к функции должны иметь атрибуты FIXED DECIMAL(5). Такое требование, в частности, приводит к тому, что часто оказывается практически невозможным использовать константы в качестве аргументов, например приходится задавать их в двоичной системе (в нашем случае аргумент для DIG пришлось бы всегда задавать пятью десятичными цифрами).

Замечание. Можно опустить не весь атрибут ENTRY, а лишь некоторые из совокупностей атрибутов, относящихся к одному или нескольким параметрам; запятые при этом отмечают пропущенные совокупности атрибутов для соответствующих параметров. Пример такого рода атрибута ENTRY можно найти во введении.

Если опущен атрибут RETURNS, то атрибуты для возвращенного значения функции, как и в фортране, устанавливаются по правилам умолчания (по первой букве имени функции).

Областью действия для величин, описанных неявно, является самый внешний блок, содержащий такое описание (исключая те

блоки, где это же имя описано явно). Напомним, что для явно описанных величин областью действия является самый внутренний блок, охватывающий оператор явного описания величины.

Рассмотрим следующие два процедурных блока:

```
P: PROCEDURE;          Q: PROCEDURE;
   . . . . . .            . . . . . .
B: BEGIN;              C: BEGIN;
     DECLARE I;              . . . . .
     . . . . . .             . . . . .
       I=I+1;                  I=I+1;
END B;                 END C;
   I=I-1;                  I=I-1;
   . . . . . .             . . . . . .
END P;                 END Q;
```

В процедуре P имеется две переменные I, локализованные в своих блоках. В процедуре Q имеется одна переменная I, описанная неявно в обоих блоках и имеющая своей областью действия самый внешний блок — процедуру Q; изменение I в одном блоке приводит к ее изменению в другом. Таким образом, можно заметить, что при наличии внутренних блоков явное описание, в котором полностью отсутствуют все атрибуты величины, существенно отличается по своему эффекту от неявного описания этой величины.

Неявное описание величин резко осуждается современной методикой программирования; не рекомендуется и бесконтрольное использование правил умолчания.

7.2. Структуры

7.2.1. Описание структур. *Структура* является объединением данных различных типов и различным образом организованных. Структура может объединять арифметические данные со строчными, скалярные величины с массивами и другими структурами — *подструктурами*. Набор величин, составляющих структуру, указывается при ее описании, например:

```
DECLARE 1 W, 2 N CHARACTER(30),
             2 D(12)BIT(8),
             2 P,FIXED,
             2 M(15, 10) FLOAT;
```

Структура, имеющая имя W, состоит из двух скалярных величин N и P, строчной и арифметической, и двух массивов D и M, одномерного и двумерного.

Использование структур в программе позволяет в зависимости от необходимости осуществлять операции целиком над структурами

(например, операции присваивания, ввода-вывода) или над отдельными их элементами, учитывая типы элементов.

Элементами структур могут быть и другие структуры, например:

```
DECLARE 1 V, 2 F CHARACTER(15),
             2 T, 3 N CHAR(30),
                  3 D(10) BIT(8),
             2 S, 3 P FIXED(2),
                  3 M(15, 10) FLOAT;
```

Структура V содержит скалярную величину F и две подструктуры; структура T содержит скаляр N и массив D, а структура S — скаляр P и массив M.

Целые числа перед именами структур и составляющих их элементов называются *уровнями*; с их помощью задается строение структуры, то есть подчиненность составляющих ее элементов. Структура всегда имеет уровень 1; составляющим ее элементам, как правило, приписывается уровень 2; если один из элементов является подструктурой, то входящие в нее элементы обычно получают уровень на единицу больше, чем уровень подструктуры.

В PL/1 имеется возможность организовать массив, элементами которого являются структуры одинакового строения, например;

```
DECLARE 1 WM(0:9), 2 N CHAR(30),
                   2 D(12) BIT(8),
                   2 P FIXED;
```

Величина WM является *массивом структур*, на что указывает атрибут размерности, приписанный к объявленному имени структуры.

Инициализация структур или массивов структур производится путем инициализации (с помощью атрибута INITIAL) составляющих ее скалярных или массивных элементов. Причем для массива структур необходимо учитывать, что количество элементов (скалярных или входящих в массив), которые необходимо проинициализировать, увеличивается пропорционально атрибутам размерностей массивов структур (или подструктур), содержащих данную величину. Например, элементы массива структур WM инициализируются следующим образом:

```
DECLARE 1 WM(0:9), 2 N CHAR(30) INITIAL((10) (30) '#'),
                   2 D(12) BIT(8) INITIAL((120) (1)'11111111'B),
                   2 P FIXED(2) INITIAL((10) 99);
```

Имена элементов в разных структурах могут совпадать. Для описания структур, у которых имена элементов и их атрибуты совпадают с элементами других структур, можно воспользоваться ат-

рибутом LIKE. Например, описание

```
DECLARE 1 UN LIKE W,
        1 G(0 : 9) LIKE WM;
```

задает структуру UN и массив структур G с элементами, аналогичными соответственно структуре W и массиву структур WM. Атрибут размерности для WM на G не переносится, и его всегда нужно задавать непосредственно (он может и отличаться от прежнего); все остальные атрибуты элементов структуры WM переносятся на элементы структуры G.

Замечание. Имена структур и их элементов могут быть объявлены [внешними (EXTERNAL — см. 4.4); кроме того, имена структур (не подструктур) могут иметь и атрибут памяти (например, STATIC, CONTROLLED).

7.2.2. Структурные переменные и квалифицированные имена. Работа со структурами производится путем использования *структурных переменных*, которые представляют собой имена структур (ср. с массивными переменными — 8.3) или массивов структур, может быть, со списком индексов в последнем случае. Например, с учетом приведенных ранее описаний структурными переменными являются

W, V, WM, WM(K), WM(9), G(0) и т. д.

По операторам

```
GET LIST(W);
PUT DATA(WM);
PUT EDIT(V) (A(15), X(10), A(30), SKIP,
                    (10) B(12), SKIP, F(5), SKIP, (150) E(16));
```

производится ввод данных для структуры W и печать значений массива структур WM и структуры V.

Для того чтобы оперировать с величинами, входящими в структуры (со скалярами, массивами и подструктурами), необходимо указывать и имя величины и имя содержащей ее структуры (структур); имена соединяются точкой. Например, переменные

W.N, W.D(1), V.T, V.T.N, V.S.M, WM(K).D(J) и т. п.

обозначают соответственно скаляр N, входящий в структуру W; элемент массива D, входящего в ту же структуру; подструктуру T, входящую в структуру V; скаляр N, входящий в структуру T, содержащуюся в V; массив M, входящий в структуру V.S; j-й элемент массива D, входящего в k-ю структуру массива структур WM и т. п. Имена с точкой называются *составными* (*квалифицированными*).

Имена старших структур могут и не указываться, если в блоке программы (простом или процедурном) нет величин с совпадающими именами. Например, в нашем случае можно просто написать T, S или T.N, S.M, F; если структуры W и V описаны в разных блоках, то можно просто использовать неквалифицированную переменную

M (разумеется, если в этих блоках имя M не используется в другом смысле, например вне структур).

У квалифицированных имен индексы могут располагаться разными способами, например (см. выше):

WM(K).D(J) или WM(K,J).D или WM.D(K,J)

Последний способ индексирования называется индексированием с «непрослаиваемыми индексами» и применяется в программах без ограничений; другие способы могут использоваться не всегда.

Квалифицированные имена, характеризующие скалярные или массивные величины, могут использоваться в выражениях и операторах вместе или наряду с обычными неквалифицированными именами и константами, например (об операции || см. 8.2.2):

```
W.N = W.N || 'РУБ' || F;
IF W.D(1) THEN V.T.N = '␣';
V.S.M = 0;
```

7.2.3. Действия над структурами. Над структурами и массивами структур могут производиться различные действия, например такие, как арифметические и строчные операции, присваивание, ввод-вывод, передача в качестве аргумента процедуры. Если при этих действиях участвует несколько структур, то все они должны иметь одинаковое строение, а для массивов структур и одинаковые размерности; действия производятся над соответствующими элементами структур. В некоторых случаях допускаются действия над структурами (массивами структур) и скаляром; не допускаются действия над структурой и массивом структур. Если атрибуты элементов, над которыми выполняются действия, различаются, то производится обычное приведение атрибутов.

Структуры и массивы структур могут входить в выражения и определять, тем самым, *структурное выражение*, другими компонентами которого могут быть только скалярные величины. Структурные выражения могут являться правыми частями операторов присваивания, в списке левой части которого находятся соответствующие по организации структурные переменные.

Используя приведенные ранее описания структур, можно написать:

```
V.S = V.S + 1; V.S = 7; WM = G;
```

Значение элементов структуры S увеличивается на единицу; все элементы структуры S получают значение 7; массиву структур WM присваиваются значения массива структур G.

Если, кроме того, дано описание

```
DECLARE 1 PR, 2 A CHAR(10),
              2 B(1 : 10) BIT(15),
```

```
1 SC, 2 E FLOAT,
      2 F(15,10) FIXED;
```

то возможны, например, следующие операторы:

```
SC=V.S; PR=PR !! V.T;
```

Структурные выражения могут являться аргументами, а имена структур — параметрами для процедур, составленных программистом; имеется только одна встроенная функция, аргументом которой может быть структурное выражение — это STRING (см. 9.1.3). Структурные параметры и аргументы должны быть одинаковым образом организованы. Если соответствующие элементы параметра и аргумента имеют разные атрибуты, то необходимо задать описание входа и в нем указать атрибуты структуры-параметра. Атрибутом структуры считается полная совокупность атрибутов всех элементов, включая и уровни. Например, если структуры PR и SC являются параметрами некоторой процедуры SDVIG, то описание входа этой процедуры будет иметь вид

```
DECLARE SDVIG ENTRY(1, 2 CHAR(10), 2 BIT(15),
                    1, 2 FLOAT, 2 (15, 10) FIXED);
```

Последовательность ввода-вывода элементов структур зависит от порядка их описания и с учетом того, что элементы многомерных массивов структур, как и обычных массивов, располагаются «по строкам». Например, для описания

```
DECLARE 1 X, 2 P(3), 2 Q(3), 1 Y(3), 2 S, 2 T;
```

по оператору

```
PUT LIST(X, Y);
```

печать будет производиться в следующем порядке:

```
X.P(1), X.P(2), X.P(3), X.Q(1), X.Q(2), X.Q(3),
Y.S(1), Y.T(1), Y.S(2), Y.T(2), Y.S(3), Y.T(3).
```

Для оператора GET LIST(X, Y) или для ввода по оператору GET EDIT элементы данных на внешнем носителе должны располагаться в указанном порядке. Для ввода по GET DATA имена вводимых данных указываются на носителе полностью квалифицированными и с непрослаиваемыми индексами, имеющими вид целых десятичных чисел.

7.3. Ситуации и их обработка

7.3.1. Ситуации. Если в ходе выполнения на ЭВМ программы, написанной на PL/1, оказывается, что в программе имеются ошибки, необнаруженные транслятором (например, результат операции вы-

ходит за установленный в машине диапазон), то говорят, что возникает *исключительная ситуация*. Ниже перечисляются основные ситуации и причины их возникновения (см. также 9.3).

OVERFLOW — переполнение; результат операции с плавающей точкой превышает по модулю 10^{75}.

UNDERFLOW — исчезновение; результат операции с плавающей точкой по модулю меньше 10^{-78} (при вычитании равных операндов не возникает).

FIXEDOVERFLOW — фиксированное переполнение; результат операции с фиксированной точкой содержит более 15 цифр (для двоичного результата — более 31 цифры).

SIZE — при присваивании (или вводе) значения с фиксированной точкой теряются старшие значащие цифры.

SUBSCRIPTRANGE — индекс выходит за границы, установленные при описании массива.

ZERODIVIDE — деление на нуль (и для плавающей и для фиксированной точки).

ENDFILE (*имя-файла*) — при вводе (по GET или READ) обнаружилось, что вводимые данные уже исчерпаны.

TRANSMIT (*имя-файла*) — устойчивая ошибка ввода-вывода.

Ситуации могут быть во включенном или выключенном состоянии. Если ситуация включена, то в случае ее возникновения в ходе выполнения программы будет осуществлена системная реакция на возникновение ситуации, состоящая обычно из печати диагностики и прекращении выполнения программы (см. 9.3). Если ситуация выключена, то системная реакция блокируется и выполнение программы продолжается обычно с неправильными результатами.

В начале выполнения программы одни ситуации включены, другие выключены. Из перечисленных ситуаций все ситуации включены за исключением SIZE и SUBSCRIPTRANGE (эти ситуации реализуются чисто программным способом и требуют много машинного времени в ходе выполнения программы).

Включение ситуации производится с помощью *префикса*, который имеет вид имени ситуации, заключенного в скобки. Префиксы (как и метки) приписываются к оператору через двоеточие, например:

```
(SIZE): R = T/V;
(SIZE, SUBSCRIPTRANGE): TEMP: B = A(I, J)*C(K);
```

Большинство ситуаций (исключая, например, ситуации ввода-вывода) могут быть выключены. Выключение ситуации осуществляется префиксом, имеющим аналогичный вид, но с добавлением частицы NO, например:

```
(NOUNDERFLOW): S = X*Y;
(NOUNDERFLOW): (NOZERODIVIDE): RETURN(A*B + C/D);
```

Действие префикса распространяется только на тот оператор, к которому он приписан. Причем приходится помнить, что в PL/1 оператором называется любая конструкция, оканчивающаяся точкой с запятой. То есть операторами являются и конструкции вида

```
DO;
DO P=1 BY 0.5 TO S;
```

Когда префикс помещается перед условным оператором, то в этом случае его действие распространяется только на выражение, находящееся между IF и THEN. Префиксы для операторов, расположенных внутри условного оператора, приписываются непосредственно к ним. Например,

```
(NOOVERFLOW): IF R*K < 0
                  THEN (SIZE): T=2*K+1
                  ELSE (SIZE): V=3*K-1;
```

Нужно отметить важный случай: префиксы могут быть приписаны и к заголовку внешней процедуры или внутреннего блока (простого или процедурного). При этом действие префикса распространяется на все операторы процедуры или блока и на вложенные в них блоки. Но действие префикса не распространяется на вызываемые процедуры, расположенные вне префиксованного блока. (Префикс перед оператором CALL контролирует только вычисление выражений в аргументах оператора.) Приписывая противоположные префиксы к операторам и внутренним блокам, можно отменить действие префикса, приписанного к внешней процедуре (или внешнему блоку), например:

```
 (SIZE):
TIP: PROCEDURE OPTIONS(MAIN);
       . . . . . .
      (NOSIZE):
     COR: PROCEDURE(X, Y, Z) RETURNS(BIT(1));
            . . . . . .
     END COR;
     . . . . .
END TIP;
```

Ситуация SIZE будет включена для всех операторов и блоков главной процедуры кроме функции COR.

Замечание. Помимо перечисленных ранее отметим здесь еще две системные ситуации: ERROR и FINISH.

Ситуация ERROR возникает в случае, когда произошла ошибка, которая не поставлена в соответствие какой-либо определенной ситуации. Ситуация FINISH возникает при выполнении оператора STOP или последнего оператора END главной процедуры. Системной

реакцией на ситуацию ERROR является ситуация FINISN; системной реакцией на ситуацию FINISH является завершение выполнения данной программы.

Системная реакция на возникновение большинства ситуаций, которая упоминалась выше, состоит в возникновении ситуации ERROR, что в свою очередь приводит к ситуации FINISH и, следовательно, к окончанию выполнения программы. Смысл такой ступенчатой реализации системной реакции на возникновение ситуаций будет понятен позже, после знакомства с оператором ON.

Форма.

префиксы ~ $\left[\left(\left\{\begin{array}{l}\textit{имя-ситуации}\\ \text{NO } \textit{имя-ситуации}\end{array}\right\}\ldots\right):\right]\ldots$

7.3.2. Оператор ON. Программист имеет возможность вместо системной задать свою реакцию на возникновение ситуаций в программе. Это осуществляется оператором ON, имеющим вид

оператор-ON ~ ON *имя-ситуации* [SNAP] {*оператор* | SYSTEM;}

Например:

```
ON OVERFLOW CALL ERROVER;
```

При возникновении ситуации OVERFLOW в каком-либо месте программы, в качестве реакции на эту ситуацию будет выполнена процедура ERROVER, написанная программистом.

В качестве оператора, находящегося внутри оператора ON и называемого ON-*единицей*, не могут быть заданы условный или составной операторы, а также ON-оператор; обычно используется блок, оператор ввода-вывода, оператор вызова или перехода.

Ключевое слово SYSTEM, заданное в операторе ON, устанавливает системную реакцию на возникновение заданной ситуации; по SNAP печатается список имен активных процедур.

С помощью операторов ON можно задать разные реакции на одну и ту же ошибку, возникающую в различных местах программы. После выполнения оператора ON его действие распространяется на все операторы процедуры, выполняемые далее, вплоть до выполнения другого оператора ON для того же имени ситуации или до выхода из блока. В частности, ключевое слово SYSTEM как раз и служит для отмены реакции, заданной программистом в предыдущем операторе ON.

Если ON-оператор находится во внутреннем блоке, то после выхода из блока восстанавливается действие ON-оператора, выполненного во внешнем блоке, а при его отсутствии восстанавливается системная реакция.

Пример.

```
SIT: PROCEDURE OPTIONS(MAIN);
     . . . . . .
     B: BEGIN; A=D/Z; . . .
          ON ZERODIVIDE P=0; . . .
            P=Q/Z; CALL EX; . . . .
     END B;
        . . . .
        S=T/Z;
        . . . . .
END SIT;
  EX: PROCEDURE;
       . . . . . .
        W=U/Z; . . .
    C:  BEGIN; . . .
          ON ZERODIVIDE GO TO LEP;
          . . . . . .
            F=X/Z; . . .
      END C;
  END EX;
```

Если при вычислении A Z равно нулю, то выполнится системная реакция на возникшую ситуацию ZERODIVIDE. Если Z равно нулю при вычислении P, то будет выполнена заданная программистом в блоке B реакция, и P получит значение нуль. После обращения к процедуре EX при вычислении W для Z, равного нулю, будет выполнена системная реакция, так как это вычисление находится уже вне блока B; это же относится и к вычислению S. Внутри блока C до выполнения оператора ON будет действовать системная реакция, а после него будет выполняться переход на метку LEP. Вне блока C действует системная реакция.

После выполнения оператора, заданного в операторе ON, происходит возврат в то место программы, где возникла ситуация, для продолжения вычислений (см. 9.3). Исключением являются ситуации ERROR и FINISH, для которых после ON-оператора продолжается выполнение системных действий: для ERROR — возникновение ситуации FINISH, а для FINISH — окончание программы. Если в ON-операторе задан оператор перехода, то для любых ситуаций вычисления продолжаются с указанной в операторе метки программы. На рис. 1 приведен детальный алгоритм обработки ситуаций PL/1. Вопросительными знаками отмечены условия, при истинности которых происходит переход по нижней стрелке, а при ложности — по боковой.

Ситуации ERROR и FINISH удобно использовать при отладке программ для реализации «аварийной» и «финальной» печати. Если задать в программе оператор вида

ON ERROR печать

то в случае возникновения какой-либо ошибки при выполнении программы произойдет печать интересующих нас величин; если ошибки в программе (или в данных) отсутствуют, то отладочная

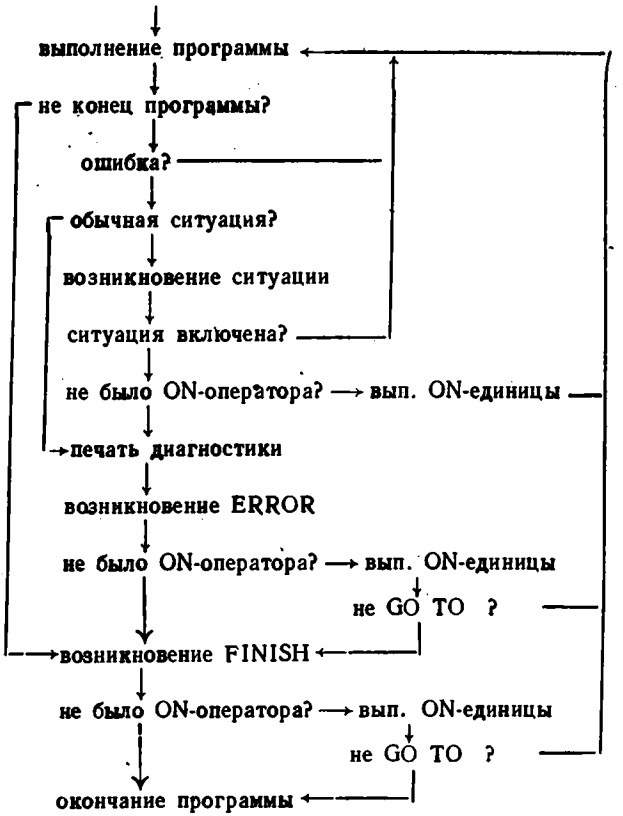


Рис. 1. Алгоритм обработки ситуаций.

печать не происходит. Поскольку значения величин печатаются в момент возникновения ошибки, это помогает программисту быстро их локализовать.

По оператору

ON FINISH печать

печать величин программы будет происходить перед завершением выполнения программы, нормальным или аварийным.

Отладочную печать можно задать любыми операторами печати, но в данном случае удобно использовать оператор вида

PUT DATA;

который выдает на печать значения всех переменных, известных в процедуре (см. 6.2). Для предупреждения некоторых неприятностей, которые могут возникнуть при использовании такого оператора, следует вместо него в операторе ON задать следующий блок:

BEGIN; ON ERROR SYSTEM; PUT DATA; END;

Замечание 1. Для искусственного возникновения некоторой ситуации используется оператор

SIGNAL *имя-ситуации*;

Этот оператор используют обычно для отладки заданной в ON-операторе реакции программиста на возникновение какой-либо ситуации (для отладки ON-единицы).

Замечание 2. Для обработки в ON-операторе возникших ситуаций могут быть использованы специальные встроенные функции, которые описаны в 9.1.5. Например, результатом выполнения функции ONLOC является символьная строка, содержащая имя процедуры, при выполнении которой возникла обрабатываемая ситуация.

По функциям ONSOURCE и ONCHAR можно узнать строку или отдельный символ, при обработке (преобразовании) которых возникла ситуация CONVERSION. Используя эти функции в качестве псевдопеременных (см. 8.4), можно исправить ошибочную строку или отдельный ее символ. Например:

```
ON CONVERSION
    BEGIN; IF ONCHAR='␣'
              THEN ONCHAR='0';
              ELSE ONSOURCE='0';
    END;
```

Если ошибочный символ при обработке строки — пробел, то он заменяется на нуль, иначе вся строка заменяется на нуль, может быть, дополненный справа пробелами. Отметим, что если символ, вызвавший ситуацию, не исправляется в ON-операторе, то при автоматически осуществляемом повторном преобразовании строки (см. 9.3.1) возникает ситуация ERROR.

7.3.3. Ситуация отладки. В PL/I введена специальная ситуация CHECK, предназначенная для использования при отладке программ. С ее помощью программист может проследить за последовательностью выполнения операторов или процедур в программе и за присваиванием значений некоторым переменным.

Включение ситуации CHECK происходит с помощью соответствующего префикса, в котором в скобках задается список имен переменных (скалярных, массивных или структурных), подпрограмм и меток операторов, интересующих программиста. Префикс CHECK может быть приписан только к блокам (простым или процедурным).

Пример.

```
(CHECK(Q,'L,'P, B)):
MULT: PROCEDURE OPTIONS(MAIN);
          . . . . . .
          GET LIST(B, N);
          Q = B**2;
          CALL P(B, Q);
      L:  N = N - 1;
          . . . . . .
END MULT;
P:    PROCEDURE(X, Y);
          . . . . . .
      X = SQRT(Y);
          . . . . . .
END P;
```

Ситуация CHECK(Q) возникает при выполнении оператора присваивания переменной Q, ситуация CHECK(B) — при выполнении оператора ввода, ситуация CHECK(L) — при выполнении оператора с меткой L, ситуация CHECK(P) — при выполнении оператора вызова P. Кроме того, при выполнении оператора вызова возникают ситуации CHECK(B) и CHECK(Q), так как этот оператор, выполняя процедуру P, может изменить значение переменных B и Q. Эти ситуации возникают только в том случае, если передача аргументов происходит именем и выход из процедуры P осуществляется через ее конец или по оператору RETURN, а не по GO TO.

Ситуации вида CHECK(P) и CHECK(L) (для подпрограмм и меток) возникают п е р е д выполнением соответствующих операторов, а остальные ситуации (при обнаружении присваивания) — п о с л е выполнения соответствующих операторов. В случае присваивания с помощью атрибута INITIAL ситуация CHECK не возникает.

В префиксе не могут фигурировать имена, являющиеся параметрами в процедуре. Например, к процедуре P нельзя приписать префикс CHECK(X) или CHECK(Y), а также префикс CHECK(B), так как переменной B в процедуре нет.

Системной реакцией на ситуацию CHECK является печать имени метки, подпрограммы или переменной, а в последнем случае еще и текущего значения переменной; печать производится каждый раз с новой строчки. Если в префиксе задано имя массива, то печатаются значения всех элементов массива. В случае, когда для какой-либо ситуации CHECK задан ON-оператор, системной печати не производится.

Замечания.

1. Те из ситуаций, которые в ОС ЕС реализованы программно (например, CHECK, SIZE, SUBSCRIPTRANGE), расходуют много времени на свою реализацию и в начале выполнения программы по умолчанию выключены. Для использования этих ситуаций их приходится включать, а по окончании отладки программы (блока) — выключать.

2. Ситуация CHECK по своим возможностям аналогична ряду отладочных режимов пакета DEBUG в фортране ЕС [16, 17].

Упражнения

7.1. Написать процедуру-функцию, которая в качестве своего значения выдает количество обращения к ней, предполагаемое менее 10^6.

7.2. Решение упражнения 4.3 оформить в виде главной процедуры, установив регулируемые границы для массива Q; значения границ ввести по оператору GET; использовать глобальные переменные.

7.3. Решение упражнения 4.4 оформить в виде простого блока, установив регулируемые границы и длину для величин B и S; использовать контролируемые переменные.

7.4. Необходимо обработать сведения об итогах сессии в 10 учебных группах по 6 предметам, представленные в следующем виде:

группа ——— предмет ———	Количество оценок			
отл	хор	удовл	неуд	нет

Вычислить общее количество указанных оценок по каждой группе и по каждому предмету, а затем по всем группам и предметам. Этапы ввода исходных данных и печати результатов опустить.

7.5. Пусть дана программа следующего строения:

```
PROG: PROCEDURE OPTIONS(MAIN);
        . . . . . .
      B: BEGIN;
        . . . . . .
      END B;
        . . . .
END PROG;
```

Вне блока B при изменении переменной X печатать текущие значения X, Y, M, а при изменении Y печатать значения всех доступных переменных; при переполнении выполнить системную реакцию. В блоке B при изменении X печатать системную диагностику, а изменения Y не фиксировать; при переполнении печатать значения всех переменных блока.

ГЛАВА 8

ДОПОЛНИТЕЛЬНЫЕ ВОЗМОЖНОСТИ

8.1. Арифметические данные

8.1.1. Операции с фиксированной точкой. Как уже говорилось ранее (см. 1.7.1), данные с фиксированной точкой характеризуются атрибутом вида

FIXED(p, q)

где p и q — целые числа, определяющие диапазон изменения и точность представления описываемых величин. Общее количество цифр в значении описываемой величины с фиксированной точкой задается числом p, а количество цифр в дробной части — числом q; таким образом, в целой части будет находиться $p-q$ цифр (может быть и нулей).

Допускается случай, когда $p < q$ или $q < 0$. Атрибуту точности (4,6) могут, например, соответствовать следующие значения: .009999, —.000001, .001234—.000472I. Первые две ($q-p=6-4=2$) цифры после запятой всегда будут равны нулю, и в памяти для их представления место не отводится.

В случае, если $q < 0$, положение точки устанавливается с п р а в а от ($p-q$)-й цифры значения. Например, атрибуту точности (4,—3) могут соответствовать следующие значения:

9999000, —0001000, 6801000— 0455000I

Последние $|q|$ цифр значения всегда будут равны нулю и в памяти машины храниться не будут.

Величины с фиксированной точкой представляются в машине абсолютно точно, как и целые величины, которые являются частным случаем фиксированных (для $q=0$). Результаты операций над величинами с фиксированной точкой, как правило, представляются с абсолютной точностью; исключением могут являться операции деления и возведения в степень.

При делении в результате сохраняется всегда 15 десятичных цифр; при возведении в степень результат может представляться и в форме с плавающей точкой (то есть с ограниченной точностью).

Ниже даются формулы, на основании которых транслятор устанавливает атрибут точности для результата операций над значениями с фиксированной точкой. С ними полезно познакомиться, для того чтобы иметь возможность избежать некоторых ошибок в программе или, по крайней мере, для того чтобы при получении неожиданных результатов в ходе отладки или счета можно было найти место ошибки.

В формулах атрибуты операндов (x_1, x_2) и результата (x) обозначаются следующим образом:

(p_1, q_1), (p_2, q_2), (p, q)

Принято также $r = p - q$.

В альтернативе $\left\{\begin{matrix}15\\31\end{matrix}\right\}$ значение 15 принимается для десятичных операндов, а 31 — для двоичных (последние описываются ниже — см. 8.1.2).

Сложение и вычитание:

$$\begin{cases} p = \mathrm{MIN}\left(1 + \mathrm{MAX}(r_1, r_2) + \mathrm{MAX}(q_1, q_2), \left\{\begin{matrix}15\\31\end{matrix}\right\}\right); \\ q = \mathrm{MAX}(q_1, q_2). \end{cases}$$

Умножение:

$$\begin{cases} p = \mathrm{MIN}\left(1 + p_1 + p_2, \left\{\begin{matrix}15\\31\end{matrix}\right\}\right); \\ q = q_1 + q_2. \end{cases}$$

Деление:

$$\begin{cases} p = \left\{\begin{matrix}15\\31\end{matrix}\right\}; \\ q = p - r_1 - q_2. \end{cases}$$

Возведение в степень:

$$\begin{cases} p = (p_1 + 1) * x_2 - 1; \\ q = q_1 * x_2; \end{cases}$$

Последние формулы верны только в случае, если показатель степени (x_2) является целым числом (не переменной, не выражением), бо́льшим или равным нулю, и если, кроме того, по формулам получается, что $p \leqslant \left\{\begin{matrix}15\\31\end{matrix}\right\}$. В противном случае результат будет представляться в плавающей форме и его точность определяется так:

$p = p_1$, если величина x_2 имеет тип целый ($q_2 = 0$);

$p = \mathrm{MAX}(p_1, p_2)$ — в остальных случаях.

Как видно из формул, при сложении, вычитании и умножении дробная часть результата с фиксированной точкой сохраняется полностью (в том числе и незначащие нули). Но поскольку максимальное количество цифр в результате ограничено, то при многократном выполнении этих операций с фиксированной точкой обычно случается, что ненулевые старшие разряды результата уже не умещаются в отводимые для значения с фиксированной точкой 15 разрядов. В этом случае возникает ситуация FIXEDOVERFLOW — фиксированное переполнение. Например, переполнение возникает при вычислении следующих выражений:

1.111111 * 2.22222 * 3.3333

или

1.000000 * 2.00000 * 3.0000

Из формул, кроме того, следует, что если операндами являются целые числа с максимальной точностью (15 или 31), то результат также будет целый с максимальной точностью.

Для операции деления общее количество цифр в результате всегда полагается максимальному значению (15 или 31), а длина дробной части зависит от длины целой части делимого и дробной части делителя. Например, для целых операндов с максимальной точностью результат деления будет иметь ту же точность ((15) или (31)). Следствием сформулированного правила является то, что операция деления в совокупности с последующей операцией сложения, вычитания или умножения может вызвать фиксированное переполнение в самых безобидных, на первый взгляд, случаях. Например, оказывается, что в PL/1 вычисление выражений 12+1/3 или 12+1/4 приводит к аварийному прекращению выполнения данной программы (см. ниже примеры). Для операции возведения в степень весьма опасен случай, когда основанием степени является число, а показателем — целая величина (в частности, переменная). Например, результат операции 2**9 или 2**M будет иметь точность (1), и для M=9 (или M=00009) получится 5E2, а не 512.

Напомним, что вычисление атрибута точности для результатов операций происходит на стадии трансляции по атрибутам операндов, а фактическое осуществление операций над конкретными значениями и приведение результата к вычисленным ранее атрибутам производится уже при выполнении программы.

Ниже приводятся некоторые примеры на вычисление точности результата по точностям операндов. Слева дается числовой пример, справа — определение атрибута результата по атрибутам или величине (для **) операндов; если операнд или результат в плавающей форме, то дополнительно указывается атрибут FLOAT.

12345.6789 + 11.111111 → 012356.790011	(9,4) + (8,6) → (12,6)
12345.6789 — 12345.11 → 000000.5689	(9,4) — (7,2) → (10,4)
999.99 + 1.102 → 1001.092	(5,2) + (4,3) → (7,3)
12345678.9 — 1.23456789 → FIXEDOVERFLOW	(9,1) — (9,8) → (15,8)
1.2 * 5 → 006.0	(2,1) * (1,0) → (4,1)
88.88 * 1.0001 → 088.888888	(4,2) * (5,4) → (10,6)
1.23456789 * 1.0000001 → FIXEDOVERFLOW	(9,8) * (8,7) → (15,15)
1/4 → 0.25000000000000	(1,0)/(1,0) → (15,14)
12 + 0.25000000000000 → FIXEDOVERFLOW	(2,0) + (15,14) → (15,14)
0020.0 ** 2 → 000000400.00	(5,1) ** 2 → (11,2)
0020.0 ** 3 → 8.0000E3	(5,1) ** 3 → FLOAT(5)
2 ** P → 5.1200E2 для P = 9.0000E0	(1,0) ** FLOAT(5) → FLOAT(5)
2 ** N → 5E2 для N = 00009	(1,0) ** FIXED(5,0) → FLOAT(1)

Замечание. Комплексная константа, входящая в выражение, PL/1 трактуется как два операнда операции сложения или вычитания, и точность константы с фиксированной точкой определяется по приведенным выше формулам. Таким образом, точность константы

+2.3 — 6.7I

равна не (2,1), а (3,1).

Фиксированное переполнение может возникнуть и в случае, когда в выражении производятся операции над переменными целого (или фиксированного) типа. Например, при вычислении выражений вида

J — 1/6, I + 5/K, I/J * K, F + G, F * G

для многих значений целых переменных I, J, K или переменных F и G с фиксированной точкой может возникнуть переполнение, и выполнение программы прекратится.

Данные и операции с фиксированной точкой, приводящие к таким неожиданным результатам, введены в PL/1 для обеспечения коммерческих, деловых, бухгалтерских расчетов, в которых, например, принято сохранять в денежном итоге все цифры вплоть до самых младших (до копейки). Использование данных с фиксированной точкой в технических и научных расчетах, как мы видели из примеров, может привести к ряду трудностей. Данные с фиксированной точкой (не целые) могут появиться в программе на PL/1, реализующей научно-технические вычисления, в основном, по двум причинам:

1) результат деления целых величин в PL/1 всегда представляется с фиксированной точкой;

2) константы, имеющие в своем изображении точку, но не содержащие букву E, относятся к данным с фиксированной точкой (а не с плавающей, как в фортране).

При выполнении операций над целыми величинами или с фиксированной точкой можно посоветовать читателям руководствоваться следующими правилами:

1) не использовать в научно-технических расчетах переменных величин в форме с фиксированной точкой, то есть описываемых атрибутом FIXED(p,q) с $q \neq 0$;

2) когда в результате выполнения некоторой операции над величинами, представляемыми в форме с фиксированной точкой, может получиться значение, требующее для своего точного представления более 15 цифр (см. примеры выше), то один из операндов такой операции следует привести к целому или плавающему типу.

К целому типу обычно приводится результат деления двух целых операндов, входящих в выражение, которое должно иметь целый тип. Например, выражение, которое на фортране имеет вид I+5/K (см. выше), в PL/1 следовало бы заменить на

I+FIXED(5/K)

Здесь FIXED — встроенная функция, в данном случае приводящая аргумент к целому типу.

К плавающему типу нужно приводить один из целых или фиксированных операндов операции, которая входит в выражение, должное иметь плавающий тип, и которая может привести к фиксированному переполнению. Операндом такой операции обычно являются или результат деления целых операндов или константы, имеющие дробную часть. Например, выражения, которые в фортране имеют вид

I/J * K, J — 1/6.0, 1.111111 * 2.22222 * 3.3333,

в PL/1 должны получить следующий вид:

I/J * FLOAT(K) или FLOAT(I/J) * K, J — 1/6.0E0,
1.111111 * 2.22222E0 * 3.3333 или 1.111111E0 * 2.22222 * 3.3333

Переменные или выражения преобразуются к плавающему типу с помощью встроенной функции FLOAT, а константы — приписыванием порядка.

Для того чтобы минимизировать использование встроенной функции FLOAT и плавающих констант в программе, следует учитывать последовательность выполнения операций разных рангов в выражении и тот факт, что результат операции получает плавающий тип, если хотя бы один из двух операндов является плавающим.

Пример.

Предположим, что требуется запрограммировать выражение

$$V_l + \frac{5}{l} - k + \frac{l}{j}(V_k + V_j) \cdot 3{,}27448$$

Если значения вектора *V* представлены в плавающей форме, то при вычислении любого из промежуточных результатов выражения

V(I)+5/I−K+I/J * (V(K)+V(J)) * 3.27448

производимого в следующем порядке:

5/I; V(I)+5/I; V(I)+5/I−K; I/J; V(K)+V(J);
I/J * (V(K)+V(J)) * 3.27448 и т. д.

фиксированное переполнение возникнуть не может.

Если же вектор V имеет тип фиксированный (целый), то для предупреждения возможных переполнений выражение PL/1 следует записать, например, так:

V(I)+5E0/I−K+I/J * FLOAT(V(K)+V(J)) * 3.27448

Но при изменении порядка выполнения операций, например, в выражении

V(I)+5E0/I−K+I/J * 3.27448 * FLOAT(V(K)+V(J))

возникнет переполнение при умножении на 3.27448 (при I/J > 1/3.27448), и надо было бы функцию FLOAT использовать раньше, при вычислении I/J или задать 3.27448E0.

8.1.2. Двоичные данные. Арифметические величины языка PL/1 могут быть десятичными или двоичными, что устанавливается при описании величин с помощью атрибутов DECIMAL или BINARY, называемых атрибутами *основания*. Для величин, которые ранее описывались атрибутами FIXED или FLOAT по умолчанию предполагается атрибут DECIMAL.

В PL/1 ЕС величины с плавающей точкой вне зависимости от атрибута основания представляются в машине в одной и той же внутренней кодировке, то есть в двоичном виде с шестнадцатиричным порядком. Величины с фиксированной точкой имеют в ЕС ЭВМ разное представление в зависимости от их основания (см. 8.5.2).

Двоичные константы PL/1 отличаются от введенных ранее десятичных тем, что в цифровой части используются только двоичные цифры (0 и 1) и в конце числа ставится буква B; в порядке по-прежнему задаются десятичные цифры, хотя сам порядок выражает степень двойки.

Примеры.

Целые числа 2, —15, 0 в двоичном виде имеют такое представление:

10B, —1111B, 0B (точность (2), (4), (1)).

Фиксированные двоичные числа с дробной частью

.1B, —10.001B, 0.101B (точность (1,1), (5,3), (4,3))

представляют следующие значения:

0.5, —2.125, 0.625

Двоичные числа с плавающей точкой

1E—3B, 11.001E+2B, .1E5B (точность (1), (5), (1))

представляют следующие значения:

0.125E0, 12.5E0, 1.6E1

Комплексные двоичные числа записываются так:

0.101B—0.001BI, —1E—3B+1.E5BI (точность (5,3), (1))

В атрибуте точности указывается количество двоичных цифр в значении (мантиссе, дробной части). О точности комплексных констант см. замечание в 8.1.1.

Для величин с фиксированной точкой представление в двоичном виде во многих случаях более эффективно, чем в десятичном. Такие величины обычно требуют для своего представления меньше памяти, и скорость выполнения операций над ними в несколько раз выше, чем для десятичных данных. Заметим также, что с точки зрения соответствия величин фортрана и PL/1 в ЕС ЭВМ целым величинам фортрана ЕС отвечают именно двоичные целые (фиксированные) величины PL/1 ЕС, а не десятичные. Причем для двоичной точности, не превышающей 15, целое значение PL/1 располагается в 2 байтах (соответствует INTEGER*2), а для большей точности (до 31) — в 4 байтах (соответствует INTEGER*4).

Поэтому в тех программах, где большую долю вычислений составляют операции над целыми или дробными величинами без порядка и эффективность работы программы играет существенную роль, следует использовать двоичные фиксированные величины вместо десятичных. *Особенно рекомендуется использовать двоичные переменные в том случае, если они применяются в качестве индексов.*

Нужно, однако, заметить, что при этом может возникнуть ряд сложностей, которые приходится иметь в виду. Эти сложности возникают обычно тогда, когда в выражение входят величины с разными основаниями. Перед выполнением операций над операндами с разными основаниями происходит преобразование десятичного операнда в двоичную систему. При этом в PL/1 считается, что одна десятичная цифра примерно соответствует 3.32 двоичным цифрам ($3.32 \approx \log_2 x/\log_{10} x$).

Приведем для справки формулы, по которым транслятор устанавливает атрибут точности (p,q) преобразованного двоичного значения *) по атрибуту десятичного (p^0,q^0). (Функция CEIL («потолок») вычисляет минимальное целое, не меньшее аргумента.)

---

*) Эти же формулы можно использовать и для преобразования из двоичной системы в десятичную, если заменить «*3.32» на «/3.32».

p=MIN(CEIL([1+]p^0 * 3.32),h);

q=$\begin{cases} \text{CEIL}(q^0 * 3.32) & \text{при } q^0 \geqslant 0; \\ \text{CEIL}(\text{ABS}(q^0 * 3.32)) * \text{SIGN}(q^0) & \text{при } q^0 < 0. \end{cases}$

Единица в p добавляется, если преобразование осуществляется к FIXED, а не к FLOAT (о функции SIGN см. 9.1.2); h обозначает максимально допустимую точность (ср. с 7.1.4):

для FIXED — $\begin{Bmatrix} 15 \\ 31 \end{Bmatrix}$, для FLOAT — $\begin{Bmatrix} 16 \\ 53 \end{Bmatrix}$.

Для демонстрации неприятностей, которые могут возникнуть при преобразовании в двоичную систему, рассмотрим следующее выражение:

J * 0.1

Если переменная J имеет атрибуты FIXED BINARY, то в качестве двоичного эквивалента константы 0.1 будет взято двоичное число 0000.0001B, которое равно 0.0625. Таким образом, для десятичного значения, не представимого точно в двоичной системе, может произойти значительная потеря точности. Для того чтобы избежать такой потери, необходимо десятичные константы задавать с большей точностью, то есть с необходимым количеством цифр в дробной части. В нашем случае следовало бы, например, задать J * 0.10000, что было бы примерно эквивалентно J * 0.09998. Кроме того, если перейти к вычислениям с плавающей точкой, то, учитывая особенности представления в ЕС ЭВМ величин в плавающей форме (см. 1.7.1 и 8.5.2), можно было бы написать J * 0.1E0 и получить результат с точностью до 6 десятичных знаков. Для тех десятичных значений, которые представляются в двоичном виде точно (например, для целых чисел), никакой потери точности не возникает. Например, выражения вида

I+1, —I+25, K+0.5, L+0.0625

никакой опасности не представляют.

Как показывает следующий пример, кроме случая, когда в дробной части преобразуемого значения находится слишком мало цифр, имеется опасность и тогда, когда этих цифр слишком много.

Пусть дан заголовок цикла следующего вида:

DO I=1 BY 1 TO D/2;

Предположим, что I—целая двоичная переменная с точностью (31,0), а десятичная переменная D имеет атрибуты FIXED DECIMAL(5,0). Точность выражения D/2, найденная по формулам п. 8.1.1, будет равна (15,10). В начале выполнения цикла, поскольку I представлено в двоичном виде, при сравнении его текущего значения с D/2 последнее должно быть приведено к двоичному виду. Двоич-

ная точность, вычисленная по приведенным ранее формулам, будет равна (31,34); таким образом, на целую часть результата деления не отводится ни одного разряда (см. 8.1.1), и после преобразования к двоичному виду результат всегда будет меньше единицы. Поэтому оказывается, что заголовок цикла задает нулевую кратность цикла, и операторы, составляющие тело цикла, не выполнятся ни разу. Например, если D=80, то D/2 будет равно

00040.0000000000,

и в двоичной системе представлено 34 нулями (целая часть результата в двоичном виде будет отброшена).

Для того чтобы избежать ошибки, в приведенном заголовке цикла нужно преобразовать D или D/2 к двоичной системе и к требуемому атрибуту точности с помощью встроенной функции BINARY (см. 8.1.3), задав после TO, например,

BINARY(D/2, 15, 5) /* $p=15$; $q=5$ */

Если D (или I) представлено с плавающей точкой, то никаких проблем в данном и в других подобных случаях не возникает (так как q в формулах перевода при этом отсутствует).

Ошибок из-за перевода в двоичную систему также не возникает, если десятичное выражение находится в правой части оператора присваивания, так как атрибут точности при этом не вычисляется по приведенным ранее формулам, а принимается равным атрибуту переменной левой части оператора. Например, при выполнении оператора присваивания

I=D/2;

для I и D с теми же атрибутами и значениями, что и ранее, I получит значение равное не нулю, а 40, которое после приведения к атрибутам I (например, FIXED BINARY(31,0)) получит вид

0...0101000

8.1.3. Встроенные функции. Встроенные функции PL/1, предназначенные для работы с арифметическими данными, делятся на математические и арифметические. Ниже дается их краткий обзор, а описание функций приведено в 9.1.1 и 9.1.2.

Математические функции. Математические функции предназначены для вычислений над величинами, представленными в форме с плавающей точкой. Результат всегда представляется с плавающей точкой; аргументы, имеющие форму с фиксированной точкой, преобразуются к плавающей форме. Таким образом, результат всегда получается с ограниченной точностью. Основные математические функции PL/1, совпадающие со стандартными функциями фортрана, были перечислены в 1.6, а на особенность их использования было указано в 2.1.3. Здесь даются некоторые дополнения.

В PL/1 имеются тригонометрические функции, аргументы которых задаются в градусах: COSD, SIND, TAND. Помимо натурального и десятичного логарифма можно вычислить и логарифм по основанию 2 с помощью функции LOG2. Многие функции определены и для комплексных аргументов.

Примеры.

SIND(45.00) ⟶ 7.071E−1, но напомним (см. 2.1), что
SIND(45) ⟶ 7.0E−1;
SIN(1.0000+2.0000I) ⟶ 3.1657E0+1.9596E0I

Из всех встроенных функций только имена математических встроенных функций могут быть аргументами при обращении к процедурам (см. 5.3.2).

Арифметические функции. Аргументы и значение этих функций, как и для математических функций, имеют арифметический тип; в отличие от математических функций они могут быть и в форме фиксированной точкой. Как правило, арифметические атрибуты (FIXED или FLOAT, DECIMAL или BINARY, REAL или COMPLEX, точность) значения функции совпадают с теми атрибутами аргумента (или основного аргумента, если их несколько), которые не оговариваются особо при описании функции. Многие функции определены и для комплексных величин.

Среди встроенных функций имеются функции для осуществления преобразования данных к заданному масштабу (FIXED и FLOAT), основанию (DECIMAL, BINARY), моде (REAL, COMPLEX), точности (PRECISION); некоторые преобразования осуществляются к необходимой точности, указываемой в аргументах. Например:

FIXED(2.79E+1, 5, 2) ⟶ 027.90
BINARY(13.25, 8, 3) ⟶ 01101.010
REAL(7.3−4.5I) ⟶ 7.3
COMPLEX(−814, 7.7E−2) ⟶ −8.14E2+7.70E−2I
PRECISION(58.76, 5, 1) ⟶ 0058.7

Функция FIXED используется, например, для приведения значений с плавающим масштабом к целому типу. Например,

FIXED(−1.79E1) ⟶ −17 (точнее −00017)

Функция ADD, MULTIPLY, DIVIDE служат для вычисления результата арифметических операций (+, *, /) с требуемой точностью, например:

ADD(13.35, 87.65, 8, 4) ⟶ 0101.0000
MULTIPLY(1.26, 3E0, 2) ⟶ 3.7E0
DIVIDE(1E2, 3, 7) ⟶ 3.333333E1

Функции FLOOR («пол»), CEIL («потолок»), TRUNC («отсекать») предназначены для получения целых значений; но для аргументов

с плавающей точкой результат также получается с плавающей точкой, а не с фиксированной, как при использовании функции FIXED.

Например:

FLOOR(4.7) ⟶ 4, CEIL(4.7) ⟶ 5, TRUNC(4.7) ⟶ 4;
FLOOR(—4.7) ⟶ —5, CEIL(—4.7) ⟶ —4, TRUNC(—4.7) ⟶ —4

Точнее надо было бы написать 04, 05, —05, —04. Кроме того,

FLOOR(4.7E0) ⟶ 4.0E0
FLOOR(4.7652E0) ⟶ 4.0000E0

Функции CONJG и IMAG позволяют получить сопряженное значение или выделить мнимую часть (в виде действительного числа). Например:

CONJG(—3—8I) ⟶ —3+8I
IMAG(—3+8I) ⟶ 8

Округление значения можно производить функцией ROUND, например:

ROUND(56.789, 1) ⟶ 56.800
ROUND(56.789, —1) ⟶ 60.000

Функция MOD определена и для отрицательного аргумента:

MOD(24, 5) ⟶ 4, MOD(—24, 5) ⟶ 1

(в фортране ЕС в последнем случае получается —4).

Функция ABS определена и для комплексного аргумента:

ABS(4—3I) ⟶ 5

Имеется функция SIGN, определяющая знак числа:

SIGN(—4.2) ⟶ —1, SIGN(4E—3) ⟶ 1, SIGN(0) ⟶ 0

8.2. Строчные данные

Строчные данные (битовые и символьные), описываемые с помощью атрибутов BIT и CHARACTER(CHAR), были охарактеризованы в 1.7.2. Здесь будут даны некоторые дополнения.

8.2.1. Изменяемая длина. Строчные данные подразделяются на величины *с постоянной длиной* (рассмотренные ранее) и *с изменяемой длиной*; последние характеризуются атрибутом VARYING. Например:

DECLARE ZAG CHARACTER(20) VARYING;

Атрибут длины в данном случае характеризует не постоянную, а максимально допустимую длину для описываемой переменной. Текущая длина переменной ZAG будет определяться по длине последнего значения, присвоенного этой переменной с помощью оператора присваивания, ввода, атрибута INITIAL и т. п. До первого присваивания длина переменной с изменяемой длиной равна нулю.

Пример.

```
DECLARE S CHAR(5) INITIAL('IBM'),
        SV CHAR(8) VARYING INITIAL('EC␣ЭВМ'),
        B BIT (6) INITIAL('1011'B),
        VB BIT(3) VARYING INITIAL('1'B);
* S='IBM␣␣', SV='EC␣ЭВМ', B='101100'B, VB='1'B */
  SV=S; /* SV='IBM␣␣' */
  VB=B; /* VB='101'B */
```

8.2.2. Строчные операции. Над строчными величинами (символьными и битовыми) могут производиться различные действия, для чего в PL/1 используются встроенные функции, перечисленные в 8.2.3 и 9.1.3, а также операция сцепления.

Операция сцепления (||), обозначаемая в PL/1 EC двумя восклицательными знаками (!!), производит присоединение второго операнда справа к первому операнду и образование новой строки с длиной, равной сумме длин строк; в фортране 77 операция сцепления имеет вид //.

Например:

'101'B !! '11010'B ⟶ '10111010'B

'IBM' !! '/' !! '370' ⟶ 'IBM/370'

Если один из операндов имеет изменяемую длину (VARYING), то и результат будет иметь этот атрибут. Например,

```
DECLARE RES CHAR(20) VARYING INITIAL('ДВИГАТЬ'),
        PR CHAR(3) INITIAL('НЕ'),
/* RES='ДВИГАТЬ', PR='НЕ␣' */
RES=PR !! RES !! '!'; /* RES='НЕ␣ДВИГАТЬ!' */
```

Если оба операнда битовые, то и результат битовый; если один из операндов символьный, то производится преобразование битового операнда к символьному виду (см. 8.2.3) и результат получается символьный. К символьному виду могут преобразовываться и арифметические данные, в частности, целые (см. 9.2.4).

Пример.

```
PUT EDIT ('ВЫПОЛНИЛСЯ' !! KOL !! '␣РАЗ') (A);
```

Перед сцеплением целая величина KOL будет преобразована к символьному виду (при этом, слева добавляется несколько пробелов), и в результате для KOL=5 напечатается, например:

ВЫПОЛНИЛСЯ␣␣␣5␣РАЗ

8.2.3. Строчные преобразования. В PL/1 допускается преобразование битовой строки в символьную и наоборот. Строчные преобразования осуществляются, например, в том случае, когда в правой и левой частях оператора присваивания находятся строчные вели-

чины разного вида. Кроме того, при операциях сцепления и отношения, если операнды являются строчными величинами разного вида, то битовая строка преобразуется к символьному виду.

Битовая в символьную. Каждый бит строки преобразуется в символ, например:

'01011'B ⟶ '01011'

Символьная в битовую. Символы 0 и 1 из строки преобразуются в биты. Если в строке присутствуют другие символы, то возникает ситуация CONVERSION (при присваивании ситуации не возникнет, если неверный символ отсекается). Например:

'01011' ⟶ '01011'B
'0101B' ⟶ ошибка

8.2.4. Строчные функции. Обработка строчных данных может производиться и с помощью встроенных функций.

Преобразование строк осуществляется функциями BIT и CHAR:

BIT('10') ⟶ '10'B
BIT('1101*59', 3) ⟶ '110'B
CHAR('1'B) ⟶ '1'

Текущую длину строки, описанную в частности с VARYING можно узнать с помощью функции LENGTH:

LENGTH('ОТВЕТ␣—␣''ДА''') ⟶ 12
LENGTH(RES) ⟶ 11 (см. 8.2.2)

Функция SUBSTR выделяет требуемую часть строки:

SUBSTR(RES, 3, 8) ⟶ '␣ДВИГАТЬ'
SUBSTR(RES, 4) ⟶ 'ДВИГАТЬ'

В фортране 77 для тех же целей используется понятие подцепочки, например, RES (3, 8).

Функции REPEAT, STRING, LOW, HIGH предназначены для образования строк из необходимых строчных компонент. Например, для EL='—**—' получим

REPEAT(EL, 2) ⟶ '—**——**——**—'

Компонентами для функции STRING являются все элементы заданного массива (или структуры). Например, для описания

DECLARE BM(2, 2) BIT(3)
INITIAL('011'B, '100'B, '001'B, '101'B);

получим

STRING(BM) ⟶ '011100001101'B

Функция BOOL позволяет выполнить любую из 16 возможных в принципе логических операций, которая задается третьим

аргументом. В примерах заданы логические операции $\equiv$ и $\supset$:

BOOL('101011'B, '011001'B, '1001'B) ⟶ '001101'B
BOOL('101011'B, '011001'B, '1101'B) ⟶ '011101'B

Функции INDEX и VERIFY используются для установления факта вхождения строки или символов, ее составляющих, в другую строку. Например, для

STR = 'NO, TONE, ON'

получим

INDEX(STR, 'ON') ⟶ 5, INDEX(STR, 'NOT') ⟶ 0

— строка 'ON' входит в STR, начиная с 5-го символа, а строка 'NOT' в STR не содержится;

VARIFY('NOT', STR) ⟶ 0, VARIFY('NEIN', STR) ⟶ 3

— все символы строки 'NOT' входят в STR, но 3-й символ строки 'NEIN' не содержится в STR.

Функция TRANSLATE используется для замены некоторых символов в строке на другие. Например:

TRANSLATE('ПАПА', 'М', 'П') ⟶ 'МАМА'
TRANSLATE('ПАПА', 'ЯД', 'АП') ⟶ 'ДЯДЯ'
Для WHOD = '+,.' и WIHOD = '␣.,' получим
TRANSLATE('+12, 345.67', WIHOD, WHOD) ⟶ '␣12.345,67'

Строки 'П' и 'М', 'АП' и 'ЯД', WHOD и WIHOD задают таблицу преобразования.

Функция UNSPEC преобразует аргумент к битовой строке, которая содержит двоичное представление аргумента в ЭВМ (в нашем случае ЕС ЭВМ). Например:

UNSPEC('IBM') ⟶ '110010011100001011010100'B
I B M
UNSPEC(—12389) ⟶ '00010010000111000100111101'B
1 2 3 8 9 —

8.3. Массивные переменные, сечения

8.3.1. Массивные переменные и выражения. В PL/1 наряду со скалярными переменными, с которыми мы имели дело ранее, имеются *массивные переменные* (*переменные над массивами*). В то время как скалярная переменная представляется в программе именем скаляра (простая переменная) или именем массива со списком индексов (переменная с индексами), массивная переменная представляется одним лишь именем массива. Значение массивной переменной характеризуется не одной константой, как для скалярной переменной, а со-

вокупностью констант; их количество соответствует количеству элементов в массиве.

Массивные переменные могут использоваться в выражениях в тех же случаях, что и скалярные переменные. Когда в выражение в качестве операнда входит хотя бы одна массивная переменная, то выражение является *массивным (выражением над массивами)*. Если операндами некоторой операции являются массивные переменные, то операция выполняется над всеми соответствующими элементами двух массивов; при этом предполагается, что операнды-массивы имеют одинаковые атрибуты размерности (граничные пары). Если один из операндов — скалярная величина, то операция производится над скаляром и всеми элементами массива. Например, для описания

```
DECLARE (A, B) (1:20) FIXED(7,6),
        (C, D) (1:20, -5:+5) FLOAT(8);
```

выражения

```
A-B, B+1, C**D, D/2
```

являются массивными.

Массивные переменные и выражения могут быть аргументами встроенных и невстроенных функций PL/1. Только для встроенных функций результат может представлять собой массивное значение (являющееся совокупностью констант). Например, значения указателей функций SIN(A—B) и SQRT(D/2) представляются соответственно совокупностью 20 и 220 констант (см. описание).

Массивное выражение может являться правой частью оператора присваивания; в этом случае в левой части оператора должна находиться массивная переменная (может быть, и список переменных) с тем же атрибутом размерности. Например:

```
A=B; C=C**D-SQRT(D/2);
```

Все элементы массива A получают значения соответствующих элементов массива B; массив C получает значение выражения, стоящего в правой части. Возможен случай, когда массивной переменной присваивается значение скалярного выражения, стоящего в правой части, например:

```
B=0; C,D=1E-5;
```

Массив B обнуляется; массивам C и D присваивается значение константы.

8.3.2. Сечения. Частным случаем массивных переменных являются *сечения*, которые представляют собой специальные части массивов. Для выделения части массива в качестве одного или нескольких индексов переменной ставится звездочка, которая имеет смысл индекса, пробегающего все значения от нижней до верхней границ,

указанных в атрибуте размерности. Над сечениями производятся те же действия, что и над обычными массивными переменными (некоторые исключения даются по ходу дальнейшего изложения). Например:

```
DECLARE (P, Q) (40, 40) FLOAT, T(40) FIXED(10,8);
. . . . . .
  T=P(1, *); P(I, *)=Q(*, J)-Q(*, K);
```

Массиву T присваиваются значения 1-й строки матрицы P, i-й строке матрицы P присваивается разность j-го и k-го столбцов матрицы Q.

Ограничение. Сечения для массивов структур не допускаются.

8.3.3. Встроенные функции для массивов. Следующие встроенные функции PL/1 предназначены для работы с массивами.

Функции SUM, PROD, ANY, ALL вычисляют сумму, произведение, логическую сумму или логическое произведение всех элементов массив . Например, для одномерных массивов A и B указатель функции

```
SUM(A*B)
```

вычисляет их векторное произведение $\left(\sum_i a_i b_i\right)$.

Указатель функции

```
ANY(C > 1E-6)
```

получает значение *истина*, если в массиве C (любой размерности) имеется хоть один элемент, больший 10^{-6}. Указатель функции

```
ALL(C > 1E-6)
```

имеет истинное значение, если все элементы массива C больше 10^{-6}.

Функция POLY может служить для вычисления значения полинома по задаваемому массиву коэффициентов и значению переменной. Например, при

```
DECLARE A(4) FLOAT INITIAL(9.3, 8.01, -6.27, 4.4),
/* ПОЛИНОМ 9.3+8.01*X-6.27*X**2+4.4*X**3 */
        Z FLOAT(6);
```

и для Z=0.1 получим

```
POLY(A, Z) ⟶ 1.00427E1
```

Функции LBOUND, HBOUND, DIM позволяют узнать текущие значения граничных пар или «протяженность» интересующей программиста размерности для указанного массива. Например, для

```
DECLARE M(-5:+5, 1:10) FIXED(4);
```

получим

LBOUND(M, 2) ⟶ 1, HBOUND(M, 1) ⟶ 5,
DIM(M, 1) ⟶ 11, DIM(M, 2) ⟶ 10

Эти функции удобно применять в процедурах для определения указанных характеристик входных параметров-массивов.

8.4. Псевдопеременные

Указатели некоторых встроенных функций могут использоваться в качестве переменных в левой части оператора присваивания (а также в качестве параметра цикла или элемента ввода оператора GET). Такие указатели функций называются *псевдопеременными*, так как им при выполнении соответствующего оператора как бы присваивается заданное значение (например, правой части оператора присваивания). Фактически принимает значение переменная (иногда их бывает и две), которая является аргументом функции-псевдопеременной. Указатель функции характеризует способ приема значения или, может быть, определяет принимающее поле в аргументе. Аргументами псевдопеременных могут быть и переменные-массивы, но не могут быть псевдопеременные.

Псевдопеременные описаны вместе с соответствующими встроенными функциями (см. 9.1). Ниже даются простейшие примеры использования псевдопеременных.

Псевдопеременные IMAG и REAL используются для изменения мнимой или действительной частей значения комплексной переменной. Например, если CV = −1+2I, то по операторам присваивания

IMAG(CV) = −5; REAL(CV) = 9;

переменная CV получит следующие значения:

CV ⟶ −1−5I и CV ⟶ 9+2I

Псевдопеременная COMPLEX дает возможность разделить комплексное значение на составные части, представляемые действительными величинами. Например, по оператору

COMPLEX(A, B) = CV;

для первоначального значения CV получаем

A ⟶ −1, B ⟶ 2

Псевдопеременная SUBSTR позволяет заменить некоторую часть строки. Например, для

STR = 'ABCDEFGH'

по оператору

SUBSTR(STR, 5, 2) = 'OS';

получаем

STR ⟶ 'ABCDOSGH'

По оператору

SUBSTR(STR, 5) = 'OS';

заменяется конец строки:

STR ⟶ 'ABCDOS␣␣'

С помощью псевдопеременной STRING можно всем элементам массива (или структуры — см. 7.2) присвоить значения компонент строки, «разобрав» для этого заданную строку на части, соответствующие по длине элементам массива (структуры). Например, при

DECLARE BM(2, 2) BIT(3);

по оператору

STRING(BM) = '011100001101'B;

получаем

BM(1, 1) ⟶ '011'B, BM(1, 2) ⟶ '100'B,
BM(2, 1) ⟶ '001'B, BM(2, 2) ⟶ '101'B

Псевдопеременная UNSPEC позволяет арифметической или строчной переменной с помощью битовой строки присвоить значение, заданное в виде конкретного представления данных на ЭВМ (см. 9.1.3). Например, при

```
DECLARE H CHAR(2),
        PACK FIXED(3),
        IR BINARY FIXED(6, 3);
```

по операторам		получим
	O C	
UNSPEC(H)	= '1101011011000011'B;	H ⟶ 'OC'
UNSPEC(PACK)	= '000100110110110 1'B;	PACK ⟶ −136
UNSPEC(IR)	= '0000000000010110'B;	IR ⟶ 010.110B

8.5. Обрабатываемые данные

Арифметические и строчные данные образуют в PL/1 единый класс данных, которые названы *обрабатываемыми* (*проблемными*). В классе обрабатываемых данных помимо описанных ранее преобразований введены преобразования *типа* — между арифметическими, битовыми и символьными данными.

Благодаря преобразованиям типа в PL/1 оказывается возможным производить арифметические операции над строчными данными и, наоборот, строчные операции над арифметическими данными. Таким образом, можно сказать, что в PL/1 нет ни арифметического, ни ло-

гического (строчного) выражения, а лишь выражение *над обрабатываемыми (проблемными) данными*.

8.5.1. Преобразования типа. Ниже кратко рассматриваются 4 преобразования типа; более подробные сведения приведены в 9.2, куда и следует обращаться по мере необходимости.

Как мы увидим позднее (п. 9.2.3), преобразования типа могут давать весьма неожиданные результаты, и поэтому, по возможности, их следует избегать, тем более, что они (как, впрочем, и остальные преобразования) сильно снижают эффективность программы. Именно, исходя из этого, во всех предыдущих главах не упоминалась возможность преобразования типа данных. Преобразования типа иногда приходится использовать в том случае, когда над одними и теми же данными необходимо производить операции различного типа: арифметические, строчные, логические.

Битовый в арифметический. Битовая строка преобразуется в положительное целое двоичное число (которое затем может быть подвергнуто необходимым арифметическим преобразованиям). Например:

'01011'B ⟶ 01011B

Арифметический в битовый. Арифметическое значение, теряя знак и дробную часть, преобразуется сначала к положительному целому двоичному числу, а затем побитно — в битовую строку. Например:

11011.01B ⟶ '11011'B

−9.85 ⟶ '1001'B

Символьный в арифметический. Символьная строка преобразуется в арифметическое значение только в том случае, если строка содержит число. Преобразование заключается в выделении числового значения, содержащегося в символьной строке. В некоторых случаях могут отбрасываться мнимая и дробная части выделенного значения; кроме того, могут производиться и другие арифметические преобразования. Например:

'␣␣−8135␣␣␣' ⟶ −8135

'␣1.3−2.4I␣' ⟶ 1

Арифметический в символьный. Преобразование арифметических данных к символьным строкам имеет специфический характер, поскольку применяется, как правило, при выводе данных на печать. Каждая десятичная цифра арифметического значения (может быть, предварительно переведенного из двоичного основания) преобразуется в соответствующий символ. Длина полученной строки зависит от атрибутов преобразуемого значения. Например:

+3126.40 ⟶ '␣␣3126.40'

−0.17350E5 ⟶ '−1.73500E+04'

8.5.2. Представление данных PL/1 в ЕС ЭВМ. В ЕС ЭВМ обрабатываемые данные PL/1 в зависимости от их атрибутов представляются следующим образом:

FLOAT DECIMAL и FLOAT BINARY — представляются в одном и том же плавающем двоичном формате. При $p \leqslant 6$ для DECIMAL или при $p \leqslant 21$ для BINARY величина занимает 4 байта, а в противном случае (при $p > 6$ или $p > 21$) — 8 байт. В фортране ЕС соответствующими являются данные, описанные с REAL*4 и REAL*8.

FIXED BINARY — представляются в фиксированном двоичном формате. При $p \leqslant 15$ величина занимает 2 байта, а при $p > 15$ занимает 4 байта. В фортране ЕС соответствующими являются данные, описанные с INTEGER*2 и INTEGER*4.

FIXED DECIMAL — представляются в упакованном десятичном формате и занимают $[p/2]+1$ байт. В фортране ЕС соответствующих данных нет,

COMPLEX — представляются двумя действительными значениями и занимают удвоенное по сравнению с REAL количество байт. В фортране соответствующие данные (COMPLEX*8 и COMPLEX*16) имеются только для COMPLEX FLOAT (DECIMAL или BINARY) с одинарной ($p \leqslant 6$ или $p \leqslant 21$) и двойной ($p > 6$ или $p > 21$) точностью.

CHARACTER — каждый символ строки занимает 1 байт. В фортране ЕС аналогичных данных нет, но символьная информация может быть размещена в данных любого типа (обычно в INTEGER).

BIT — в одном байте размещается от одного до 8 битов (см. ниже 8.5.3). В фортране ЕС соответствующими данными можно считать логические (LOGICAL*1 и LOGICAL*4) данные. Кроме того, некоторым аналогом является 16-ричная информация фортрана ЕС, которая может быть размещена в данных любого типа.

8.5.3. Уплотнение данных. С помощью атрибутов ALIGNED и UNALIGNED программист может задавать «неплотное» (*выровненное*) или «плотное» (*невыровненное*) размещение величин в памяти. В первом случае ускоряется выполнение программы, но за счет дополнительных расходов памяти. Во втором случае экономится память, но замедляется выполнение программы.

По атрибуту ALIGNED переменные с атрибутами FIXED BINARY размещаются с границы полуслова (для $p \leqslant 15$) или слова (для $p > 15$), а переменные с атрибутом FLOAT размещаются, начиная с границы слова (для $p \leqslant 6$ или $p \leqslant 21$) или двойного слова. Остальные типы данных размещаются с начала байта.

По атрибуту UNALIGNED все типы данных за исключением битовых размещаются, начиная с границы байта, а битовые (в том числе и элементы массива) размещаются с очередного свободного

бита. Как видим, символьные и фиксированные десятичные данные всегда размещаются, начиная с границы байта.

По умолчанию строчные данные уплотнены, а арифметические — нет.

Размещение в фортране соответствующих данных (см. 8.5.2) установлено такое же, что и по умолчанию в PL/1 (см. 7.1.4).

Упражнения

8.1. Составить процедуру, в которой с помощью сечений вычисляется произведение прямоугольных матриц:

$$c_{ij} = \sum_{k=1}^{L} a_{ik} \cdot b_{kj}; \quad i = 1, \ldots, M; \quad j = 1, \ldots, N.$$

8.2. Составить процедуру, которая в строке, содержащей число с фиксированной точкой, заменяет дробную часть, если она есть, на пробелы включая и точку.

8.3. Составить процедуру, которая переставляет местами k первых и k последних символов в строке.

8.4. Составить процедуру-функцию, которая имеет значение **истина**, если среди соответствующих строк двух матриц нет таких, что все их элементы попарно равны; в противном случае функция получает значение **ложь**.

ГЛАВА 9

СПРАВОЧНАЯ

В этой главе собран справочный материал, посвященный встроенным функциям, преобразованиям, ситуациям. Здесь же приводятся списки ключевых слов PL/1, символов ЕС ЭВМ и синтаксических форм.

9.1. Встроенные функции

Ниже дается сводка встроенных функций PL/1, которые разделены на математические, арифметические, строчные, функции для работы с массивами и разные функции. Если не оговорено особо, то аргументами функции являются скалярные или массивные выражения; в последнем случае значением функции будет массив (если массивных аргументов несколько, то они должны иметь одинаковые границы).

Если у функции имеется несколько аргументов, над которыми при вычислении функции производятся какие-либо действия, то эти аргументы преобразуются как операнды выражений (см. 2.1.2), и функция получает результирующие атрибуты (если не указано иное).

Ниже для аргументов встроенных функций используются следующие обозначения:

x, y, z — скалярные и массивные выражения (для псевдопеременных — скалярные и массивные переменные);

l, n, p, q — целые десятичные числа;

$c = a + i \cdot b$ — комплексное значение.

К аргументам x, y, z иногда будут приписываться обозначения атрибутов, указывающих на необходимый тип аргумента. Например, xA, yT, zH обозначают арифметические, битовые, символьные аргументы или преобразуемые к ним. Если при обращении заданы аргументы с другими атрибутами, то автоматически производится преобразование к указанным атрибутам; преобразования см. в 8.2 и 9.2.

О максимальной точности h см. в 7.1.4.

Примеры обращений к встроенным функциям были даны в 8.1.3, 8.2.4, 8.3.3 и 8.4.

9.1.1. Математические функции. Аргументы всех функций имеют арифметический тип и плавающий масштаб или преобразуются к нему. Значение функции имеет плавающий масштаб; остальные атрибуты совпадают с атрибутами аргумента, в том числе атрибуты точности и размерности. Ниже x обозначает действительное или комплексное значение, а xR — только действительное.

Идентификаторы математических функций могут быть аргументами при вызове процедур.

ATAN(x)	arctg(x); главное значение в радианах; для комплексного x (кроме $x=\pm 1\cdot i$) выдается LOG((1+$i\cdot x$)/(1−$i\cdot x$)).
ATAN(xR, yR)	arctg(x/y); то есть: если $y>0$, то arctg(x/y); если $x>0$ и $y=0$, то $+\pi/2$; если $x\geqslant 0$ и $y<0$, то $\pi+$arctg(x/y); если $x<0$ и $y=0$, то $-\pi/2$; если $x<0$ и $y<0$, то $-\pi+$arctg(x/y).
ATAND(xR[,yR])	arctg(x[/y]); исключается случай 0/0; главное значение в градусах: ATAND(x[,y]) = (180/π)·ATAN(x[,y]).
ATANH(x)	arcth(x); предполагается, что ABS(x) < 1; для комплексного x (кроме $x=\pm 1+0\cdot i$) выдается LOG((1+x)/(1−x))/2.
COS(x)	cos(x); x — в радианах; для комплексного x выдается cos(a) ch(b) − $i\cdot$sin(a) sh(b).
COSD(xR)	cos(x); x — в градусах.
COSH(x)	ch(x); для комплексного x выдается ch(a) cos(t) + $i\cdot$sh(a) sin(b).
ERF(xR)	функция ошибок $\mathrm{ERF}(x)=\frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}dt$.
EXP(x)	e^x.
LOG(x)	ln(x); исключается $x\leqslant 0$; для комплексного x (кроме $x=0+0\cdot i$) выдается LOG(ABS(x)) + $i\cdot\varphi$, где $-\pi<\varphi<+\pi$ ($c=r\cdot e^{i\cdot\varphi}$).

LOG10(x)	$\log_{10}(x)$; исключается $x \leqslant 0$.
LOG2(x)	$\log_2(x)$; исключается $x \leqslant 0$.
SIN(x)	$\sin(x)$; x—в радианах; для комплексного x выдается $\sin(a)\,\mathrm{ch}(b) + i \cdot \cos(a)\,\mathrm{sh}(b)$.
IND(xR)	$\sin(x)$; x—в градусах.
SINH(x)	$\mathrm{sh}(x)$; для комплексного x выдается $\mathrm{sh}(a)\cos(b) + i \cdot \mathrm{ch}(a)\sin(b)$.
SQRT(x)	$+\sqrt{x}$; исключается $x < 0$; для комплексного x выдается главное значение: $u \pm i \cdot v$, где $u > 0$ или $u = 0$ и $v \geqslant 0$.
TAN(x)	$\mathrm{tg}(x)$; x—в радианах.
TAND(xR)	$\mathrm{tg}(x)$; x—в градусах.
TANH(x)	$\mathrm{th}(x)$.

9.1.2. Арифметические функции. Значение функции имеет арифметический тип. Аргументы x и y имеют тип арифметический или преобразуемый к нему. Если не оговорено особо, то те атрибуты значения функции, которые не упоминаются, совпадают с атрибутами основного обрабатываемого аргумента. Аргументы функций могут быть действительными или комплексными (x), только действительными (xR) или только комплексными (xC).

ABS(x)	Результат: $\lvert x \rvert$; для комплексного x выдается $+\sqrt{a^2+b^2}$; точность для FIXED COMPLEX: (MIN(h, $p+1$), q).
ADD(x, y, p[,q])	Сложение $x+y$; результат имеет заданную точность ($p \leqslant h$); для FIXED q обязательно.
BIN[ARY](x[,p[,q]])	Преобразование в двоичное основание, с указанной (иначе см. 8.1.2) точностью.
CEIL(xR)	Минимальное целое, не меньшее x; точность для FIXED: (MIN(h, MAX ($p-q+1$, 1)), 0).
{COMPLEX / CPLX} (xR, yR)	Образование $x+i \cdot y$; может использоваться как псевдопеременная для разделения комплексного значения на составные части (они становятся вещественными).

CONJG(*xC*)	Сопряженное значение: $a-i\cdot b$.
DEC[IMAL](*x*[,*p*[,*q*]])	Преобразование в десятичное основание с указанной (иначе см. 8.1.2) точностью.
DIVIDE (*x*, *y*, *p*[,*q*])	Деление x/y с указанной точностью результата ($p \leqslant h$); для FIXED q обязательно.
FIXED(*x*[,*p*[,*q*]])	Преобразование к фиксированному масштабу. Когда p и q не заданы, то они берутся по умолчанию: 5,0 или 15,0.
FLOAT(*x*[,*p*])	Преобразование к плавающему масштабу. Когда p не задано, то оно берется по умолчанию: 6 или 21.
FLOOR(*xR*)	Максимальное целое, не большее x; точность для FIXED: (MIN(h, MAX($p-q+1$, 1)), 0).
IMAG(*xC*)	Мнимая часть комплексного значения в виде вещественного значения; может использоваться как псевдопеременная для изменения мнимой части значения комплексной переменной.
MAX(*xR*,*yR*,..)	Максимальное из заданных (не менее двух) значений; если заданы массивы, то результат—массив. Точность для FIXED: (MIN(h, $\max\limits_i(p_i-q_i)+\max\limits_i(q_i)$), $\max\limits_i(q_i)$).
MIN(*xR*, *yR*,..)	Минимальное из заданных (не менее двух) значений; далее, как в MAX.
MOD(*xR*, *yR*)	Положительный остаток деления x на y; то есть минимальное положительное число z, такое, что $(x-z)/y$ является целым. Может возникнуть ситуация ZERODIVIDE, а для FIXED может быть и SIZE (см. 9.3.1). Точность для FLOAT: (MAX(p_1, p_2)); для FIXED: (MIN(h, p_2-q_2+MAX(q_1, q_2)), MAX(q_1, q_2)).
MULTIPLY(*x*,*y*,*p*[,*q*])	Умножение x на y с указанной точностью результата ($p \leqslant h$); q при FIXED обязательно.
PREC[ISION](*x*,*p*[,*q*])	Преобразование в указанную точность.
REAL(*xC*)	Взятие действительной части x; может использоваться как псевдопеременная

Для замены действительной части значения комплексной переменной.

ROUND(x, n) — Округление. Для FIXED производится на n-й цифре (справа от точки при $n \geqslant 0$ или слева при $n < 0$); отрицательные значения округляются по абсолютной величине. Точность для FIXED: (MIN($p+1, h$),q). Для FLOAT устанавливается в единицу самый правый (31-й или 63-й) двоичный разряд конкретного представления; строчные данные не меняются.

SIGN(xR) — Знак числа. Значение, равное 1 или -1, или 0, имеет атрибуты FIXED BINARY (15, 0).

TRUNC(xR) — Отсечение дробной части:

$$\text{TRUNC}(x) = \begin{cases} \text{FLOOR}(x), & \text{при } x \geqslant 0, \\ \text{CEIL}(x), & \text{при } x < 0; \end{cases}$$

точность для FIXED:
(MIN(h, MAX($p+1-q$,1)),0).

9.1.3. Строчные функции. Основными аргументами всех функций, если не оговорено противное, являются строчные выражения или преобразуемые к ним. Некоторые аргументы могут быть только битовыми (xT), только символьными (xH) или только целыми (xI), а также преобразуемыми к ним. Аргументом функции UNSPEC является любое проблемное (обрабатываемое) выражение (xO).

BIT(xT[,l]) — Преобразование x к битовой строке с заданной (иначе см. 9.2.2) длиной.

BOOL(xT, yT, zT) — Первые четыре бита строки $z(z_1, z_2, z_3, z_4)$ задают логическую (булевскую) операцию над соответствующими битами строк x и y. Результирующие биты:

$$v_i = \begin{cases} z_1, & \text{для } x_i = 0 \text{ и } y_i = 0; \\ z_2, & \text{для } x_i = 0 \text{ и } y_i = 1; \\ z_3, & \text{для } x_i = 1 \text{ и } y_i = 0; \\ z_4, & \text{для } x_i = 1 \text{ и } y_i = 1. \end{cases}$$

Перед операцией длины x и y выравниваются. Каждый из аргументов может быть массивным выражением.

CHAR(x[,l]) — Преобразование к символьной строке с заданной (иначе см. 9.2.4) длиной.

HIGH(l)	Результат: «максимальная» символьная строка с длиной, равной l; в ЕС ЭВМ каждый символ будет кодирован в 16-ричной системе как FF.
INDEX(x, y)	Определение номера символа в x, начиная с которого y первый раз входит в x; если y в x не входит — выдается нуль. Атрибуты результата: FIXED BINARY (15, 0).
LENGTH(x)	Результат: длина строки x в виде FIXED BINARY (15, 0).
LOW(l)	Результат: «минимальная» символьная строка с длиной, равной l; в ЕС каждый символ будет кодирован как 00 (в 16-ричном виде).
REPEAT(x, n)	Сцепление x с собой n раз. Длина результата: $(n+1)\cdot$LENGTH(x). Если $n \leqslant 0$, то x будет результатом (может быть преобразованным).
STRING(v)	Сцепление в одну строку всех элементов строчной переменной v, которая предполагается массивной или структурной (для скалярной v она же и будет результатом). Все элементы структуры должны быть строками одного типа; сечение массива не может быть задано в аргументе. Функция используется и как псевдопеременная. В этом случае присваиваемое скалярное значение «разбирается» при присваивании на части, соответствующие элементам v. Если строки не хватает, то оставшиеся элементы будут пустыми (для VARYING) или заполнятся пробелами.
SUBSTR(x, y[,z])	Выделение из строки x подстроки, начинающейся с позиции y и оканчивающейся на позиции $y+z-1$ (или на последней позиции, если z не задано). Подстрока имеет атрибут VARYING и длину, равную z (или $l-y+1$), где l — длина строки x. Выделяемая подстрока должна вся находиться внутри строки x, в противном случае

аргументы исправляются автоматически. Сигнализация о неверно заданных аргументах будет производиться, если включена ситуация STRINGRANGE (см. 9.3.1).

Функция используется и как псевдопеременная для изменения части строки x; если строка x имеет атрибут VARYING, то длина строки при ее изменении не устанавливается, то есть такой строке ранее уже должно быть присвоено какое-либо значение.

TRANSLATE ($xH,yH[,zH]$) — Посимвольный перевод x по таблице: z — вход, y — выход (y выравнивается по z). Если z не задано, то в ЕС предполагается строка длиной 256, содержащая символы, закодированные по возрастанию от 00 до FF в 16-ричной системе.

UNSPEC($x0$) — Внутреннее представление значения проблемного выражения в виде битовой строки. Длина строки зависит от атрибутов аргумента:

для FIXED DECIMAL(p, q) равна 8*FLOOR(($p+q$)/2);

для FIXED BINARY(p, q) равна 16 (при $p \leqslant 15$) или 32 (при $p > 15$); для констант всегда равна 32;

для FLOAT DECIMAL(p) равна 32 ($p \leqslant 6$) или 64 ($p > 6$);

для FLOAT BINARY(p) равна 32 ($p \leqslant 21$) или 64 ($p > 21$);

для CHARACTER(l) равна 8*l;

для BIT(l) равна l;

для COMPLEX(p) равна двойной длине действительного значения.

Функция используется также и как псевдопеременная. При этом присваиваемое значение преобразуется к битовой строке с длиной, зависящей от атрибутов переменной-аргумента (см. выше), и присваивается псевдопеременной прямо в битовой «машинной» форме без приведения к ее атрибутам. Но если псевдоперемен-

ная строчная и с VARYING, то длина определяется по присвоенному значению.

VERIFY(x, y) Проверка того, что все символы x имеются в y — в этом случае выдается нуль. Иначе выдается номер первого слева символа из x, не входящего в y; атрибуты результата FIXED BINARY(15,0). Для пустого y выдается 1; но если x пустой, то всегда выдается 0.

9.1.4. Функции для массивов. Основными аргументами функций являются выражения над массивами или их сечениями; значения функций — скаляры.

ALL(xT) Все строки массива выравниваются по самой длинной (с дополнением нулями справа), после чего производится операция И (&) над соответствующими битами строк. Результат: битовая строка с длиной самой длинной строки.

ANY(xT) Так же, как в ALL, но производится операция ИЛИ (|).

DIM(x, n) Размер (протяженность) n-го измерения массива x (верхняя граница — нижняя граница $+1$); атрибуты результата: FIXED BINARY(15,0).

HBOUND(x, n) Верхняя граница n-го измерения x; атрибуты результата FIXED BINARY(15,0).

LBOUND(x, n) Нижняя граница n-го измерения x; атрибуты результата FIXED BINARY(15,0).

POLY(yA, zA) Вычисление значения полинома, образованного по одномерным массивам $y(m:n)$ и $z(k:l)$ следующим образом:

$$y_m + \sum_{j=1}^{n-m} \left(y_{m+j} \times \prod_{i=0}^{j-1} z_{k+i} \right).$$

Для $k-l-1 > m-n$ полагается $z_{k+l} = z_l$ при $k+l > l$. Если z — скаляр, то выдается значение

$$\sum_{j=0}^{n-m} y_{m+j} \cdot z^j.$$

PROD(xA) Произведение всех элементов x, преобразованных к плавающей форме; другие атрибуты сохраняются.

SUM(xA) Сумма всех элементов x, преобразованных к плавающей форме; другие атрибуты сохраняются.

9.1.5. Функции ситуаций. Нижеследующие функции предназначены только для использования в ON-операторе или в блоках, им активизированных. Все функции не имеют аргументов.

DATAFIELD — Результат: символьная строка переменной длины (≤255 в EC), содержащая поле данных, которое вызвало ситуацию NAME; для других ситуаций (не считая ERROR и FINISH после NAME) выдается пустая строка).

ONCHAR — Результат: символьная строка длиной 1, содержащая ошибочный символ, вызвавший ситуацию CONVERSION. Если функция применяется после других ситуаций, не связанных с CONVERSION, то выдается пробел. Функция, используемая как псевдопеременная, позволяет исправить неверный символ.

ONCODE — Результат: двоичное число точности (15,0), определяющее тип ошибки (тип прерывания). Таблица ошибок здесь не приводится.

ONFILE — Результат: символьная строка переменной длины (≤31 в EC), содержащая имя файла в случае, если возникла ситуация ввода-вывода или CONVERSION; для CONVERSION, не связанного с вводом-выводом, выдается пустая строка.

ONLOC — В виде символьной строки переменной длины выдается имя входа в процедуру, в которой возникла обрабатываемая ситуация.

ONSOURCE — Результат: символьная строка переменной длины (≤255 в EC), содержащая ошибочную строку, вызвавшую ситуацию CONVERSION. Если функция применяется после других ситуаций, не связанных с CONVERSION, то выдается пустая строка. Функция, используемая как псевдопеременная, позволяет заменить обрабатываемую строку (она будет подвергнута обработке после выхода из ON-оператора); заменяющее выражение правой части оператора присваивания преобразуется к символьной строке с длиной, ранее преобразуемой строки.

9.1.6. Разные функции.

ADDR (переменная) — **Для размещенной базированной переменной выдается адрес размещения в памяти (значение соответствующего указателя), для неразмещенной — значение функции не определено.**

ALLOCATION (переменная)	Для управляемой переменной, размещенной в памяти, выдается '1'B, для неразмещенной — '0'B.
COUNT (файл)	Выдается количество элементов данных, переданных при выполнении последнего GET или PUT; атрибут результата: FIXED BINARY(15,0).
DATE	Выдает символьную строку длины 6 с датой в виде ггммдд (год, месяц, день).
LINENO(SYSPRINT)	Результат: номер текущей печатаемой линии с атрибутами FIXED BINARY (15,0). Номер, равный нулю, устанавливается по PAGE.
NULL	Служит для присваивания указателю пустого значения (типа адреса), используемого как признак неразмещенности переменной в памяти.
TIME	Выдает символьную строку длины 9, содержащую текущее время дня в виде ччммссттт (час, минута, секунда, миллисекунды).

9.2. Преобразования типа

Ниже приводится детальное описание преобразований типа, имеющихся в PL/1; все преобразования упоминались ранее (см. 8.5).

Отметим, что следует различать два случая выполнения преобразований: а) при операции присваивания и б) при вычислении выражений. В первом случае атрибуты, к которым производится преобразование, заданы явно атрибутами переменной левой части оператора присваивания. («Операция присваивания», кроме того, выполняется в операторах DO, RETURN, GET, по INITIAL и при передаче фактических параметров по значению.) Во втором случае атрибуты определяются в зависимости от вида операции и атрибутов операндов. Напомним, что из арифметических преобразований, при этом, возможны только следующие (см. 2.1.2):

REAL ⟶ COMPLEX, FIXED ⟶ FLOAT, DECIMAL ⟶ BINARY

Ниже при указании точности для преобразованного значения будет предполагаться именно этот 2-й случай, если не оговорено обратное. Атрибуты преобразуемой величины будем называть *атрибутами источника*, а атрибуты результата преобразования или операции — *атрибутами мишени (цели)*; некоторые атрибуты мишени могут определяться и форматом при редактируемом вводе-выводе.

Заметим также, что определение атрибутов мишени производится на стадии трансляции, и при этом осуществляется вставка в оттранслированную программу подпрограмм преобразования (и выдача преду-

преждающей диагностики). Фактическое преобразование конкретных значений к атрибутам мишени осуществляется на стадии выполнения оттранслированной программы. Преобразование констант производится на стадии трансляции, и диагностика обычно не выдается.

Ниже p^0, q^0 и l^0 используются для обозначения точности и длины до преобразования, а p, q и l — после преобразования значения; h обозначает максимально допустимую точность (см. 7.1.4).

9.2.1. Битовый в арифметический. От битовой строки берется не более 31 разряда справа (или $\leqslant 56$, если требуется преобразование к FLOAT), и они приводятся к атрибутам FIXED BINARY(31,0) (или FLOAT BINARY(31) — при этом отсекаются 25 младших разрядов). Если среди отсекаемых старших разрядов встречаются не нулевые, то возникает ситуация SIZE. Таким образом, всегда $p=31$.

Примеры.

'1011'B ⟶ +0...01011B (слева 27 нулей)

'10101...1'B ⟶ +11...1B (31 единица)

32 единицы

''B ⟶ +0...0B (31 нуль)

9.2.2. Арифметический в битовый. Арифметическое значение преобразуется сначала к целому положительному значению с атрибутами FIXED (p^0-q^0, 0) для значения с фиксированной точкой или к FIXED (p^0, 0) для значения с плавающей точкой, а затем переводится в двоичную систему (для DECIMAL). Далее каждая двоичная цифра преобразуется в бит. При этом длина строки равна: $l=p^0[-q^0]$ — для двоичных значений или $l=\mathrm{CEIL}((p^0[-q^0])*3.32)$ — для десятичных значений. При $p^0-q^0\leqslant 0$ получается пустая строка.

Примеры.

+1011.010B ⟶ '1011'B

−12.75 ⟶ '0001100'B

1.275E1 ⟶ '000000000001100'B

9.2.3. Символьный в арифметический. Преобразование символьной строки к арифметическому значению заключается в выделении числового значения из строки. Это возможно только в случае, если строка содержит правильную числовую константу (с точностью $p^0\leqslant h$); иначе возникает ситуация CONVERSION. Пустая строка преобразуется в нуль. Атрибуты, которые получает выделенное после преобразования значение, не зависят от его первоначальных атрибутов, а зависят только от атрибутов мишени, то есть в данном случае от атрибутов взаимодействующей с ним (через какую-либо операцию) величины.

При операции присваивания находящееся в строке значение получает атрибуты переменной, стоящей в левой части оператора.

При арифметических операциях мнимая часть выделенного из строки значения всегда отбрасывается. Выделенное действительное значение может получить следующие атрибуты:

FLOAT — если атрибут мишени FLOAT. Получаемая точность зависит от основания мишени: она равна 15 для DECIMAL или 53 для BINARY;

FIXED — если атрибут мишени FIXED. Получаемая точность зависит от основания мишени и полагается равной (15,0) для DECIMAL или (31,0) для BINARY, то есть дробная часть выделенного значения отбросится.

Если мишень также имеет атрибут CHARACTER или если выполняется одноместная операция, то операнды приводятся к FIXED DECIMAL(15,0).

Пример.

```
DECLARE (H1, H2) CHAR(12) INITIAL('1.5E0+1.2E0I'),
        (CF1, CF2) FIXED(2,1) COMPLEX INITIAL(2.4+3.7I),
        (E1, E2) FLOAT(2) INITIAL(4.8E0),
        B1 FIXED BINARY(4,2) INITIAL(2.25);
```

Ниже приведены выражения, полученные значения и точность.

Выражения	Значения	Точность
CF1+H1	3.4+3.7I	(15,1)
B1+H1	3.25	(31,2)
E1+H1	6.3E0	(15)
−H1	−1	(15,0)
H1+H1	2	(15,0)

Для операторов

```
CF2=H1; , E2=H1; , H2=H1;
```

получим

```
1.5+1.2I, 1.5E0, '1.5E0+1.2E0I'
```

с точностью или длиной (2,1), (2), (12).

9.2.4. Арифметический в символьный. Арифметическое значение преобразуется к символьной *числовой* строке, которая будет содержать десятичное число, равное по величине преобразуемому, и, может быть, окаймленное пробелами. Данное преобразование используется, в основном, при выводе арифметических значений на печать.

Длина и конкретный вид полученной строки зависит от атрибутов преобразуемого значения.

Плавающее в символьный. Значение преобразуется к виду десятичного числа с плавающей точкой в формате

{␣|−}ц.ц...Е{±}цц

и располагается в конце строки. Мантисса имеет p^0 цифр, первая из которых не равна нулю (для ненулевого числа); точка ставится

после первой цифры. Количество цифр в порядке всегда равно двум. Длина строки: $l = p^0 + 6$.

Примеры:

$$.003445E0 \longrightarrow \text{'␣3.44500E}-03\text{'}$$
$$-389E4 \longrightarrow \text{'}-3.89E+06\text{'}$$
$$110001E1B \longrightarrow 98 \longrightarrow \text{'␣9.8E}+01\text{'}$$

Фиксированное в символьный. *Случай* $p^0 \geqslant q^0 \geqslant 0$. Значение преобразуется к виду десятичного числа с фиксированной точкой и располагается в конце строки. Незначащие нули (кроме нуля перед точкой) и знак плюс заменяются пробелами; точка в случае $q^0 = 0$ не ставится. Длина строки: $l = p^0 + 3$.

Примеры.

$$5 \longrightarrow \text{'␣␣␣5'}$$
$$-024.97 \longrightarrow \text{'␣␣}-24.97\text{'}$$
$$.00 \longrightarrow \text{'␣0.00'}$$
$$1.1100B \longrightarrow 1.75 \longrightarrow \text{'␣␣1.75'}$$

Случай $p^0 < q^0$ *или* $q^0 < 0$. Значение преобразуется к виду:

{␣|—}ц...F{±}ц [ц [ц]]

и располагается в конце строки. Целое десятичное число (ц...) содержит p^0 цифр. Одна, две или три цифры в конце служат для изображения порядка числа, полагаемого равным $-q^0$ ($-128 \leqslant -q^0 \leqslant 127$). Буква F в середине числа имеет тот же смысл, что и буква E перед порядком числа. Длина строки: $l = p^0 + 3 + k$, где k — количество цифр в q^0.

Примеры.

$$.003445 \longrightarrow \text{'␣3445F}-6\text{'}$$
$$-3890000 \longrightarrow \text{'}-389F+4\text{'}$$
$$-089000 \longrightarrow \text{'␣}-890F+3\text{'}$$

Комплексное в символьный. Комплексное значение преобразуется к символьному виду десятичного комплексного числа. Действительная и мнимая части (со знаками) преобразуются отдельно по только что рассмотренным правилам, но полученное мнимое число (с буквой I в конце) располагается в строке сразу вслед за действительной частью. Причем старшие незначащие нули не заменяются пробелами, а уничтожаются, и остальные цифры мнимой части сдвигаются влево (внутри полученного комплексного числа не может быть пробелов). Длина строки: $l = 1 + 2l_R$, где l_R — длина строки для соответствующего действительного значения.

Примеры.

$$-18.3E1+0.19E-3I \longrightarrow \text{'}-1.83E+02+1.90E-04I\text{'}$$
$$12.24-00.07I \longrightarrow \text{'␣␣12.24}-0.07I\text{␣␣'}$$
$$15I \longrightarrow \text{'␣␣␣␣0}+15I\text{␣␣'}$$

9.3. Ситуации

Ситуация	Условие возникновения	Включена?	Выключается?	Реакция системы	Возврат после ON-оператора
9.3.1. Ситуации вычислений.					
CONVERSION	При вводе или преобразовании поток или строка имеет неправильный вид, например, содержит недопустимый символ или незаконное число.	Да	Да	Результат не определен; диагностика; ERROR.	На повторение преобразования (если символ не был исправлен, то — ERROR).
FIXEDOVERFLOW	Значение с фиксированной точкой имеет цифр более 15 (для DECIMAL) или 31 (для BINARY).	Да	Да	То же.	На следующую операцию.
OVERFLOW	Значение с плавающей точкой по абсолютной величине не ϵ больше допустимого (10^{75}).	Да	Да	То же.	То же.
SIZE	При присваивании, преобразовании или вводе теряются старшие значащие разряды значения (для FIXED).	Нет	Да	То же.	То же.
STRINGRANGE	Неверное задание аргументов при обращении к функции SUBSTR.	Нет	Да	Исправление аргументов; повторение.	См. реакцию системы.
SUBSCRIPTRANGE	Значение индекса выходит за заданные границы.	Нет	Да	Результат не определен; диагностика; ERROR.	На следующую операцию.

Продолжение

Ситуация	Условие возникновения	Включена?	Выключается?	Реакция системы	Возврат после ON-оператора
UNDERFLOW	Абсолютный результат с плавающей точкой меньше допустимого (10^{-78}). При вычитании равных чисел не возникает.	Да	Да	Результат — нуль; диагностика; продолжение вычислений.	На следующую операцию.
ZERODIVIDE	Деление на нуль.	Да	Да	Результат не определен; диагностика; ERROR.	То же.
9.3.2. Ситуации ввода-вывода.					
ENDFILE (файл)	Вводимые данные исчерпались.	Да	Нет	Диагностика; ERROR.	На следующий оператор.
ENDPAGE (SYSPRINT)	Страница исчерпалась (при печати).	Да	Нет	Печать с новой страницы (с 1-й строчки).	Продолжение выполнения оператора вывода.
KEY (файл)	Ошибочный номер региона в операторе.	Да	Нет	Диагностика; ERROR.	На следующий оператор.
NAME (файл)	Имя во входном потоке (для DATA), неизвестное в блоке или отсутствующее в списке данных (см. функцию DATAFIELD).	Да	Нет	Диагностика; пропуск ошибочного поля данных.	На ввод следующего поля данных.
RECORD (файл)	Несоответствие длины записи длине переменной в операторе ввода-вывода.	Да	Нет	Диагностика; ERROR.	На следующий оператор.
TRANSMIT (файл)	Устойчивая ошибка при вводе или печати данных.	Да	Нет	Диагностика; ERROR.	Продолжение ввода-вывода

Продолжение

Ситуация	Условие возникновения	Включена?	Выключается?	Реакция системы	Возврат после ON-оператора
UNDEFINEDFILE (файл)	Невозможность открытия файла из-за ошибок в OPEN, DD или из-за несоответствия атрибутов.	Да	Нет	Диагностика; ERROR.	На следующий оператор.
9.3.3. Ситуации отладки.					
CHECK(*i*,..)	Присваивание переменной с именем *i* или выполнение оператора (процедуры) с меткой (именем) *i*.	Нет	Да	Печать *i* и его значения (для переменной); на продолжение вычислений.	На продолжение вычислений.
CONDITION(*i*)	По оператору SIGNAL *i*; (*i* считается EXTERNAL).	Да	Нет	Сообщение (печать *i*) и продолжение вычислений.	На следующий оператор.
9.3.4. Ситуации системной реакции.					
ERROR	Какая-либо ошибка в программе (в частности см. выше) или SIGNAL ERROR;.	Да	Нет	Возникновение ситуации FINISH.	См. реакцию системы (при GO TO в ON-операторе — продолжение вычислений)
FINISH	Ошибка (см. ERROR), STOP; или SIGNAL FINISH; в программе, RETURN; или END; для главной процедуры.	Да	Нет	Окончание программы.	См. реакцию системы (при GO TO в ON-операторе или для SIGNAL FINISH; — продолжение вычислений).

9.4. Синтаксические формы

9.4.1. Сравнительные синтаксические формы.

Фортран ЕС	Пункты
Символы и комментарий	
буква-латинская~{ A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z }	1.1
буква~{*буква-лат* | ▢ }	
цифра~{0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9}	1.1
спецзнак~{+ | — | * | / | = | . | , | (|) | ' | & }	1.1
символ-языка~{ *буква* | *цифра* | *спецзнак* }	1.1
символ-ЭВМ~{ *символ-языка* | Б | Г | Д | Ж | З | И | Й | Л | П | У | Ф | Ц | Ч | Ш | Щ | Ь | Ы | Э | Ю | Я | < | > | ¬ | ! | : | _ | ␣ | % | ? | [|] | # | @ }	1.1
комментарий~С *символ-ЭВМ* ... <С — в 1-й позиции строчки бланка>	1.8
Метка, переменная, функция	
имя~*буква* [*буква* | *цифра*] ... <до 6 символов>	1.2
метка~*цифра* ... <до 5 цифр>	1.4
переменная~ { *имя-скаляра* <переменной> | *имя-массива* (*индекс* ,..) }	1.5
индекс~*выражение-целое* <положительное>	
функция <*указатель функции*>~ ~*имя-функции* (*аргумент* ,..)	1.6
аргумент~{*выражение* | *имя-переменной* | *имя-процедуры*}	5.3

Специальными значками отмечены понятия (конструкции) языков, отличающиеся по своим возможностям:
■ — новое понятие PL/1; □ — отсутствующее в PL/1 понятие; ● — обладает дополнительными возможностями по сравнению с фортраном; ○ — имеются ограничения по сравнению с фортраном; + — дополнительные возможности см. в 9.4.2.

PL/1 ЕС

Символы и комментарий

буква-латинская ~ {A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z}

буква ~ {*буква-лат* | ¤ | # | @} ●

цифра ~ {0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9}

спецзнак ~ {+ | − | * | / | = | . | , | (|) | ' | < | > | & | ! | : | ; | _ | ␣ | %} ●

символ-языка ~ {*буква* | *цифра* | *спецзнак*}

символ-ЭВМ ~ {*символ-языка* | Б | Г | Д | Ж | З | И | Й | Л | П | У | Ф | Ц | Ч | Ш | Щ | Ь | Ы | Э | Ю | Я | ? | [|]}

комментарий ~

~ /* *символ-ЭВМ* ... <кроме сочетания */> */ <в любых позициях строчки (строчек) бланка> ●

Метка, переменная, функция

идентификатор ~ *имя* ~ *буква* [*буква* | *цифра* | _] ... <до 31 символа> ●

имя-внешнее ~ *буква-лат* [*буква-лат* | *цифра*] ... <до 7 символов>

метка ~ *идентификатор* <:>

переменная ~ { *имя-скаляра* | *имя-массива* (*индекс* ,..) } +

индекс ~ *выражение-целое* ●

указатель-функции ~ *имя-функции* [(*аргумент* ,..)]

аргумент ~ {*выражение* | *имя-переменной* | *имя-процедуры* | *метка* | *переменная-меточн* | (*аргумент*)} ●

Фортран ЕС	Пункты
Константы	
константа ~ {*константа-арифметическая* | *константа-логическая* | *константа-символьная* | *константа-шестнадцатиричная*}	1.7
константа-арифм ~ *число* ~ {*число-цел* | *число-вещ* | *число-компл*}	1.7.1
число-целое ~ [±] *целое-без-знака* *целое-без-знака* ~ *цифра* ...	1.7.1
число-вещественное ~ ~ { [±] (*цифра* ...).(*цифра* ...) [±] (*цифра* ...) [.] (*цифра* ...) { D | E } [±] *цифра* [*цифра*] }	1.7.1
число-комплексное ~ (*число-вещ*, *число-вещ*)	1.7.1
константа-логическая~{.TRUE. | .FALSE.}	1.7.2
константа-литеральная <*символьная*> ~ ~{'{*символ-ЭВМ* <кроме'> | ''} ...' | *целое-без-зн* H *символ-ЭВМ* ...}	1.7.2
константа-шестнадцатиричная ~ ~Z{*цифра* | A | B | C | D | E | F} ...	1.7.2
Выражения	
выражение~{*выражение-арифм* | *выражение-логич*}	
выражение-арифметическое~ ~[+ | −] { *константа-арифм-без-зн* *переменная-арифм* *указатель-функции-арифм* (*выражение-арифм*) } {+ | − | * | / | * *} ..	2.1

PL/1 EC

Константы

константа~{***константа-арифметическая***
| ***константа-битовая*** | ***константа-символьная*** | <*нет*> | ***меточная***}

константа-арифм~*число*~{*число-действит* | *число-компл*}
число-действительное~{*число-цел* | *число-фикс* | *число-плав*} +
число-целое~[±] *целое-без-знака*
целое-без-знака~*цифра* ...
■ ***число-фиксированное***~[±] (*цифра* ...) [.] (*цифра* ...)
число-плавающее~*число-фикс* E [±] *цифра* [*цифра*] ○

число-комплексное~
~{*число-действ* {±} *число-действ-без-зн* I | *число-действ*. I}
константа-логическая~{'1'B |'0'B} <см. *конст-битовая*>
константа <*строка*>-*символьная*~
~[(*число-целое*)]'{*символ-ЭВМ* <кроме'>| ''}...' ●
константа <*строка*>-*битовая*~
~ [(*число-целое*)] '{1 | 0}...'B <см. кодировку в 9.7>

Выражения

выражение~{*выраж-арифм* | *выраж-логич* | *выраж-символьное*} +
выражение-арифметическое~

$$\sim\left[\begin{matrix}+\\-\end{matrix}\right]\left\{\begin{matrix}\textit{константа-арифм-без-зн}\\ \textit{переменная-арифм}\\ \textit{указатель-функции-арифм}\\ \textit{выражение-арифм}\end{matrix}\right\}\{+\mid-\mid*\mid/\mid**\}..$$

Фортран ЕС	Пункты
выражение-логическое~ ~ { [.NOT.] { *константа-логич* | *переменная-логич* | *указатель-функции-логич* | *отношение* | (*выражение-логич*) } } {.AND. | .OR.} ..	2.2
отношение~*выраж-арифм* { .LT. | .LE. | .GT. | .GE. | .EQ. | .NE. } *выраж-арифм*	2.2 8.2
Операторы *оператор*~[*метка*] *оператор-без-метки*	1.4 7.3.1
Операторы (без меток) *оператор-присваивания*~ ~ { *переменная-арифм* = *выражение-арифм* | *переменная-логич* = *выражение-логич* | ASSIGN *метка* TO *переменная-цел-спец-без-инд* }	3.1

PL/1 EC

выражение-битовое~

~ { [¬] { *константа-битовая* | *переменная-битовая* | *указатель-функции-бит-стр* | *отношение* | (*выражение-битовое*) } } {& | ! | !! } .. ●

отношение~*выражение* { < | < = | > | > = | = | ¬ = } *выражение*

■ *выражение-символьное*~

~ { *константа-симв* | *переменная-симв* | *указатель-функции-симв* | (*выражение-симв*) } {!!} ..

Операторы

оператор~[*префикс*:] ... [*метка*:] ...

оператор-без-меток-и-префиксов ●

■ *префикс*~ ({ *имя-ситуации* | NO *имя-ситуации* } ,..)

Операторы (без меток и префиксов) +

оператор-присваивания~

~ { *переменная-арифм* ,.. = *выражение-арифм*; | *переменная-логич* ,.. = *выражение-логич*; | *переменная-меточн* ,.. = {*метка* | *переменная-мет*}; } ● +

Фортран ЕС	Пункты
оператор-перехода~	3.2
~GO TO { *метка* / (*метка* ,..), *переменная-целая* / *переменная-меточ-спец*, (*метка* ,..) }	
оператор-вывова~ CALL *имя-п/п* [(*аргумент* ,..)]	3.3
аргумент~{*выражение* | *имя-переменной* | *имя-процедуры* | & *метка*}	5.3
оператор-продолжения~CONTINUE	3.4
оператор-условный ~	3.5
~ { IF (*выраж-действит*) *метка*, *метка*, *метка* <<0, =0, >0> / IF (*выраж-логич*) *оператор* <кроме IF, DO> }	
оператор-составной ~ <нет>	3.6
оператор-цикла ~	3.7
~ DO *метка переменная-целая* = *спецификация-цикла* [*оператор* <с новой строчки>] ...	
метка оператор <кроме DO, GO TO, IF <A>, RETURN, STOP,...>	
спецификация-цикла ~	
~ {*конст-цел* | *перем-цел*}, {*конст-цел* | *перем-цел*} [, {*конст-цел* | *перем-цел*}] <*конст* | *перем* >0>	
оператор-паузы <связи с пультом> ~	3.9
~ PAUSE [*целое-без-зн* | *константа-симв*]	
оператор-окончания ~STOP [*целое-без-зн*]	3.10
Описания (спецификации)	
описание-явное ~ *тип* [*длина*] {*имя* [*длина*] [*размерность*] [*нач-знач*]} ,..	4.1

PL/1 EC

оператор-перехода ~
~ GO TO { *метка* | *переменная-меточ-с-индексом* | *переменная-меточ-без-индексов* } ;
<списки меток — в описании переменной>
оператор-вызова ~ CALL *имя-п/п* [(*аргумент* ,..)];
аргумент ~ {*выражение* | *имя-переменной* | *имя-процедуры* | *метка* | *переменная-меточ* | (*аргумент*)} ●
оператор-пустой ~ ;
оператор-условный ~
~ { <см. 3.5> | IF (*выраж-логич*) THEN *оператор* [ELSE *оператор*] } ●

■ *оператор-составной* ~ DO; *оператор* ... END;
оператор-цикла ~
~ DO *переменная-арифм* = *спецификация-цикла*; ●
оператор ...
END;

спецификация-цикла <упрощенная> ~
~ *выраж-арифм* TO *выраж-арифм* [BY *выраж-арифм*] ● +

оператор-связи-с-пультом ~
~ DISPLAY (*константа-симв*) [REPLAY (*переменная-симв*)]; ●
оператор-окончания ~ STOP; ○

Описания

описание-явное <упрощенное> ~
~ DECLARE { {*имя* [*размерность*] [*тип*] [*точность* | *длина*] [*нач-знач*] [прочие]} ,... | ({*имя* [*размерность*] [*нач-знач*]} ,..) *тип* [*точность* | *длина*] } ; ● +

Фортран ЕС	Пункты
тип ~ {INTEGER | REAL | COMPLEX | LOGICAL}	4.2
длина ~ * { {4 | 2} | {4 | 8} | {8 | 16} | {4 | 1} } <для соответствующего типа>	
равмерность ~ ({ *переменная-цел* <для параметров> | *целое-без-знака* } ,..)	4.2
нач-значение ~ <см. также DATA, BLOCK DATA> ~/{[*цел-без-зн* *] *константа*} ,../	4.3
описание-размерности ~ DIMENSION {*имя-массива* (*целое-без-зн*)} ,..	
описание-общих-переменных ~ COM ON {[/*имя-блока*/] {*имя-перем* [*равмерность*]} ..	4.4
описание-эквивалентных-переменных ~ ~ EQUIVALENCE {(*переменная* ,..)} ,..	4.5
описание-аргументов-процедур ~ ~ EXTERNAL *имя-процедуры* ,..	
Процедур	
процедура-подпрограмма ~ ~ SUBROUTINE *имя-процедуры* [({*параметр* | *}],. .)] *описание* ... *оператор* ... <на отдельных строчках> END	5.2 5.3 5.4
процедура-функция ~ ~*тип* FUNCTION *имя-процедуры* [*длина*] (*параметр* ,..) *описание* ... *оператор* ... <на отдельных строчках> END	5.2 5.3 5.4
оператор-присваивания-значения-функции ~ ~ *имя-функции* = *выражение*	
оператор-возврата ~ RETURN [*целое-без-зн* <для метки>]	5.2
оператор-входа ~ ENTRY *имя-входа* [(*параметр* ,..)]	
параметр ~ {*имя-переменной* | *имя-процедуры* | /*имя-переменной*/}	

PL/1 EC

тип ~ {FIXED | FLOAT | COMPLEX | BIT | CHARACTER} ●

точность ~ (*целое-без-зн* [, *число-цел*])
длина <строки> ~ (*выражение-целое*) ●
размерность ~
~ { ({[*выраж-цел*:] *выраж-цел*} ,..) <для непараметров> | (*,..) <для-массивов-параметров> } ●
нач-значения <упрощенные> ~
~ INITIAL ({[(*цел-без-зн*)] *константа*} ,..) +
☐ *описание-размерности* ~
~ <нет—см. описание-явное>
описатель-общих-<внешних>-переменных ~ EXTERNAL

описатель-совмещаемых-переменных ~
~ DEFINED *имя-переменной* [(*индекс* <м. б. с iSUB> ,..) | POSITION (*целое-без-зн*)] ● ●
☐ *описание-процедур-аргументов* ~ <нет>

Процедуры

процедура-подпрограмма ~
~ [*префикс*:] ... {*имя-проц*:} ... PROCEDURE [(*параметр* ,..)] [RECURSIVE] ; ●
{*описание* | *оператор*} ...
END [*имя-процедуры*] ;
процедура-функция ~ [*префикс*:] ... {*имя-процедуры*:} ...
PROCEDURE [(*параметр* ,..)] [RECURSIVE] ●
RETURNS (*тип* {*точность* | *длина*}) ;
{*оператор* | *описание*} ...
END [*имя-процедуры*];
оператор-присваивания-значения-функции~<нет—см. RETURN>
оператор-возврата ~ RETURN [(*выражение*) <для функции>];
оператор-входа ~ {*имя-входа*:} ... ENTRY [(*параметр* ,..)]
[RETURNS (*тип* {*точность* | *длина*})]; ●
параметр ~ {*имя-переменной* | *имя-процедуры*} ○
■ *описание-входа* ~ DECLARE *имя-процедуры*
ENTRY [(*описатели-параметра* ,..)]
[RETURNS (*тип* {*точность* | *длина*})];

Фортран ЕС	Пункты
Форматный ввод-вывод	
оператор-ввода-вывода ~	6.3.1
~ { READ / WRITE } (*файл*, *ссылка-на-формат*) *элемент-в-в* ,..	
файл ~ {*целое-без-зн* | *переменная-цел*}	6.1
ссылка-на-формат ~ {*метка* | *имя-массива*}	
элемент-ввода ~ {*переменная* | *элемент-DO*}	
элемент-вывода ~ {*переменная* | *элемент-DO*}	
элемент-DO ~ (*элемент* ,.. , *переменная* = *специф-цикла*)	
оператор-формата ~	
~ *метка* FORMAT (*спецификация-форматов*)	
спецификация-форматов ~	6.3.4
~ ([*повторитель*] *формат* / [*повт*] ({[*повт*] *формат*} ,..) / [*повт*] ({[*повт*] ({[*повт*] *формат* <для компл>} ,..)} ,..)) ,..	
повторитель ~ *целое-без-знака* <> 0>	
Форматы	6.3.2
ф-для-целых ~ I*w*	
ф-для-фикс-точки ~ [*k*P <без повт>] F*w*.*d*	
ф-для-плав-точки ~ [*k*P <без повт>] E*w*.*d*	
ф-для-дв-точности ~ [*k*P <без повт>] D*w*.*d*	
ф-универсальный ~ [*k*P <без повт>] G*w*.*d* □	
ф-для-симв ~ A*w*	
ф-для-логич ~ L*w* *ф-для-16-ричных* ~ Z*w* □	
ф-пропуска ~ *n*X <без повт>	6.3.3
ф-позиции ~ T*n* <без повт>	
ф-прогона ~ / <без повт>	
ф-литеральн ~ *конст-литер* <без повт> □	
<кроме того см. управляющие символы ␣, 0,1, +>	
{*w* | *d* | *k* | *n*} ~ *число-целое*.	

PL/1 EC

Потокоориентированный ввод-вывод с редактированием

оператор-ввода-вывода ~

~ { GET / PUT } FILE (*имя-файла*) (*опции*) ●

(EDIT {(*элемент-в-в* ,..) ({R (*метка*) | *специф-форматов*})} ...);

■ *опции-ввода* ~ [SKIP [(*n*)]) (COPY) ⟨*n* — *выражение-целое*⟩

■ *опции-вывода* ~ {SKIP [(*n*)] | LINE (*n*) | PAGE [LINE (*n*)]}

элемент-ввода ~

~ {*переменная* | *псевдо-перем* ⟨п. 8.4⟩ | *элемент*-DO}

элемент-вывода ~ {*выражение* | *элемент*-DO}

элемент-DO ~ (*элемент-в-в* ,..DO *переменная* = DO*специф-цикла*)

оператор-формата ~

~ {*метка*:} ... FORMAT (*спецификация-форматов*)

спецификация-форматов ~

~ ([*повторитель*] *формат* / [*повт*] ({[*повт*] *формат*} ,..) / [*повт*] ({[*повт*] *формат*} ,..)} ,..) / ⟨и т. д.!⟩) ●

повторитель ~ {*целое-без-знака* | (*выражение-целое*)} ●

Форматы

ф-для-целых ~ F (*w*)

ф-для-фикс-точки ~ F (*w* [,*d* [,*k*]])

ф-для-плав-точки ~ E (*w*, *d* [, *p*])

ф-для-дв-точности ~ ⟨см. формат E⟩

■ *ф-для-компл* ~ C ({*ф*-F | *ф*-E} [, {*ф*-F | *ф*-E}])

ф-для-симв ~ A [(*w*)]

ф-для-битов ~ B [(*w*)]

ф-для-пропуска ~ X (*n*)

ф-для-позиции ~ COLUMN (*n*)

ф-для-прогона ~ SKIP [(*n*)] ●

ф-для-строчки ~ LINE (*n*) ●

ф-для-страницы ~ PAGE

ф-удаленный ~ R ({*метка* | *перем-меточн*}) ●

{*w* | *d* | *k* | *n* | *p*} ~ *выражение-целое* ●

Фортран ЕС	Пункты
Именованный ввод-вывод *оператор-ввода-вывода* ~ ~ { READ / WRITE } (*файл*, *имя-списка*) *оператор-имен-списков* ~ ~ NAMELIST {/*имя-списка*/ *имя-переменной* ,..}....	6.2
Бесформатный ввод-вывод *оператор-ввода-вывода* ~ ~ { READ / WRITE } (*файл*) *элемент-в-в* ,.. *оператор-закрытия-файла* ~ ~ REWIND *файл* <см. также ENDFILE>	6.5.1
Прямой ввод-вывод *оператор-описания-файла* ~ DEFINE FILE {*файл* (*целое*, *целое*, *символ*, *перем-цел*)} ,.. *оператор-ввода-вывода* ~ { READ / WRITE } (*файл'* *выраж-цел* [,*метка*]) *элемент-в-в* ,..	6.5.2
Главная подпрограмма *описание*... <*с новых строчек*> *оператор*... <*с новых строчек*> END	5.1 7.3.1

PL/1 EC

Ввод-вывод данными

оператор-ввода-вывода ~

~ { GET / PUT } FILE (*имя-файла*) (DATA [(*элемент-в-в* ,..)]) (*опции*); ●

элемент-ввода ~ *имя-переменной*

элемент-вывода ~ {*переменная* | *элемент*-DO}

Последовательный ввод-вывод записями

оператор-ввода-вывода ~

~ { READ FILE (*имя-файла*) INTO (*имя-переменной*); / WRITE FILE (*имя-файла*) FROM (*имя-переменной*); }

оператор-закрытия-файла ~

~ CLOSE {FILE (*имя-файла*)} ,..;

■ *оператор-перезаписи* ~

~ REWRITE FILE (*имя-файла*) FROM (*имя-переменной*);

■ *оператор-пропуска* ~

~ READ FILE (*имя-файла*) IGNORE (*выраж-цел*);

Прямой ввод-вывод записями

описатель-организации-файла ~

~ ENVIRONMENT (REGIONAL (1) *характ-записи*)

оператор-ввода-вывода ~

~ { READ FILE (*имя-файла*) / WRITE FILE (*имя-файла*)

INTO (*перем*) [KEYTO (*перем-цел*) / KEY (*выраж-симв*)] ; }

FROM (*перем*) [KEYFROM (*выраж-симв*)]; }

■ *оператор-перезаписи* ~ REWRITE FILE (*имя-файла*)

FROM (*перем*) [KEY (*выраж-симв*)];

■ *оператор-стирания* ~

~DELETE FILE (*имя-файла*) KEY (*выраж-симв*);

Главная процедура

[*префикс*:] ... {*имя-процедуры*:} ...

PROCEDURE [(*параметр-симв*)] OPTIONS (MAIN); ●

{*описание* | *оператор*} ...

END [*имя-процедуры*];

9.4.2. Дополнительные формы PL/1. Приведенные выше синтаксические формы PL/1 не отражают всех возможностей PL/1, описанных в книге. В основном, были затронуты те возможности, которые имеют аналоги в фортране. Синтаксические формы фортрана, как правило, были приведены для тех конструкций, для которых можно найти соответствующие в PL/1. Приведем синтаксические формы для ряда уточненных или дополнительных конструкций PL/1.

Переменные (п. 7.2, 8.3)

переменная ~ {*перем-скалярн* | *перем-массивн* | *перем-структ*}

перем-структ ~ {*перем-структ-скалярн* | *перем-структ-массивн*}

перем-ск~{*имя-ск* | *имя-масс*(*выр-ск-цел*,..) | {*имя-стр-ск.*}...*перем-ск*}

перем-масс~{*имя-масс*
| *имя-масс* ({*выр-ск-цел* | (*) <сечение — п. 8.3.2>},..)
| {*перем-стр-ск.*}...*пер-масс* | {*пер-стр-ск.* | (*пер-стр-масс.*)}...*пер-ск*}

перем-стр-ск~[*имя-структ.* | *имя-масс-стр* (*выр-ск-цел*).]...
{*имя-структ* | *имя-масс-стр*(*выр-ск-цел*,..)}

перем-стр-масс~[*перем-стр-ск.*]...'*имя-масс-стр*

имя-переменной~[*имя-структуры.*]... [*имя-скаляра* | *имя-массива* | *имя-структуры*]

Константы (п. 8.1.2)

число-действит~{*число-действ-десятичн* | *число-действ-двоичн*}

число-действит-десятичн~<см. в 9.4.1>

число-действит-двоичн~ {*число-действит-двоичн-фикс* | *число-действит-двоичн-плав*}

число-действит-дв-фикс~[±]({0 | 1}...)[.]({0 | 1}...)B

число-действит-дв-плав~[±]({0 | 1}...)[.]({0 | 1}...)
E[±] *циф* [*циф*[*циф*]]B

Выражение (п. 7.2, 8.3)

выражение <по организации значения>~
~{*выраж-скалярн* | *выраж-масс* | *выраж-стр-ск* | *выраж-стр-масс*}

выражение-скалярн~ {[+ | − | ¬] {*перем-скалярн-обрабат* | *функция-скалярн* | *константа-обрабат* | (*выраж-скалярн*)}} *операция*..

выраж-массивн~ {[+ | − | ¬] {(*перем-масс-обраб*) | (*функция-масс*) | (*выраж-масс*) | *выраж-скалярн*}} *операция*..

выраж-структ-ск~ {[+ | − | ¬] {(*перем-структ-ск*) | (*функция-струк-ск*) | *выраж-скалярн* | (*выраж-структур-ск*)}} *операция*..

выраж-структ-масс~{[+ | − | ¬] (*перем-структ-масс* | *функция-структ-масс* | *выраж-скалярн* | *(выраж-структ-масс)*)} *операция*..

операция~{¬ | ** | * | / | + | − | !! | < | > | <= | >= | = | ¬= | ¬> | ¬< | & | !}

Операторы

оператор-присваивания <п. 7.2, 8.3, 8.4>~

~{ {*перем-ск* | *псевдоперем-ск*},.. = {*выр-ск* | *метка* | *перем-мет-ск*};
 {*перем-масс* | *псевдоперем-масс*},.. = {*выр-масс* | *выр-ск* | *перем-мет-масс* | *перем-мет-ск* | *метка*};
 {*перем-стр-ск* | *псевдоперем-стр-ск*},.. = {*выр-стр-ск* | *выр-ск* | *перем-мет-ск* | *метка*};
 {*перем-стр-масс* | *псевдоперем-стр-масс*},.. = {*выр-стр-масс* | *выр-ск* | *перем-мет-ск* | *метка*};}

спецификация-цикла <п. 3.8>~

~{*выр-ск* [BY *выр-ск-арифм* [TO *выр-ск-арифм*] | TO *выр-ск-арифм* [BY *выр-ск-арифм*]] [WHILE (*выр-бит*)]},..

параметр-цикла~{*перем-ск* | *псевдоперем-ск*}

оператор-реакции-на-ситуацию <п. 7.3>~

~ON *ситуация* [SNAP] {*оператор* <кроме IF, DO, ON> | SYSTEM;}

оператор-имитации-ситуации~ SIGNAL *ситуация*;

оператор-размещения <п. 7.1.3>~

~ALLOCATE {*имя-переменной* [*атрибут* ...]},..;

оператор-освобождения~ FREE *имя-переменной*,..;

Блоки и описания

блок <п. 7.1.1>~{*блок-простой* | *блок-процедурный*}

блок-простой~[*префикс*:]... [*метка*:]... BEGIN;
{*описание* | *оператор*}... END [*метка-блока*];

описание <п. 4.1, 7.2>~DECLARE *список-описаний*;

список-опис~{[*целое-уровень*␣] {*имя* | (*список-опис*)} [*атрибут*...]},..

атрибут-нач-значений <см. также 4.3>~ INITIAL (*нач-значения*);

нач-значения~ {[(*выраж-цел*)] {*константа* | * | (*нач-значения*)}},..

Ввод-вывод

оператор-ввода-вывода-списком <п.3.11>~

~{GET | PUT} FILE (*имя-файла*) (LIST (*элемент-в-в*,..)) (*опции*);

оператор-ввода-вывода-строкой <п.6.4.2>~

~{GET | PUT} STRING ({*перем-строчн* | *псевдоперем-строчн*}) *спецификац-данных*;

оператор-открытия-файла~

~OPEN {FILE (*имя-файла*) [*опции-и-атрибуты*]},..;

9.5. Ключевые слова PL/I

Ключевое слово	Сокращение	Перевод	Пункты
ALIGNED		выровненный	8.5.3
ALLOCATE		разместить	7.1.3
AUTOMATIC	AUTO	автоматический	7.1.3,7.1.4
BACKWARDS		назад	6.5.1,6.5.3
BASED		базированный	7.1.3
BEGIN		начало	7.1.1
BINARY	BIN	двоичный	8.1.2
BIT		битовый	1.7.2
BY		шаг	3.7,3.8
CALL		вызвать	3.3,5.2
CHARACTER	CHAR	символьный	1.7.2
CHECK		проверка	7.3.3,9.3.3
CLOSE		закрыть	6.5.1
COLUMN	COL	колонка	6.3.3
COMPLEX	CPLX	комплексный	1.7.1
CONDITION		состояние	9.3.3
CONSEQUTIVE		последовательный	6.5.3
CONTROLLED	CTL	управляемый	7.1.3
CONVERSION	CONV	преобразование	6.3.2,7.3.2, 8.4.3,9.3.1
COPY		копировать	6.1
DATA		данные	6.1,6.2
DECIMAL	DEC	десятичный	8.1.2
DECLARE	DCL	описать	4.1
DELETE		стереть	6.5.2
DEFINED	DEF	совмещаемый	4.5
DIRECT		прямой	6.5.3
DISPLAY		показать	3.9
DO		выполнить	3.6,3.7,3.8, 6.1
EDIT		редактирование	6.1,6.3
ELSE		иначе	3.5
END		конец	3.6,3.7,3.8, 5.1,7.1.1
ENDFILE		конец данных	6.1,7.3.1, 9.3.2
ENDPAGE		конец страницы	9.3.2

Продолжение

Ключевое слово	Сокращение	Перевод	Пункты
ENTRY		вход	5.2,7.1.4
ENVIRONMENT	ENV	оборудование	6.5.2
ERROR		ошибка	6.1,7.3.1, 7.3.2
EXTERNAL	EXT	внешний	4.4,7.1.4
FILE		файл	6.4.1
FINISH		окончание	7.3.1,7.3.2 9.3.4
FIXED		фиксированный	1.7.1, 8.1.1
FIXEDOVER-FLOW	FOFL	фиксированное переполнение	7.3.1, 8.1.1, 9.3.1
FLOAT		плавающий	1.7.1
FORMAT		формат	6.3.1
FREE		освободить	7.1.3
FROM		из	6.5.1
GET		ввести	3.11,6.1,6.4
GO TO	GOTO	перейти на	3.2
IF		если	3.5
IGNORE		пропустить	6.5.1
INITIAL	INIT	начальный	3.2, 4.3
INPUT		ввод	6.5.1
INTERNAL	INT	внутренний	4.4,7.1.4
INTO		в	6.5.1
KEY		ключ	6.5.2,9.3.2
KEYFROM		ключ — из	6.5.2
KEYTO		ключ — в	6.5.3
LABEL		метка	3.1.3,3.3
LIKE		подобный	7.2.1
LINE		линия	6.1,6.3.3
LIST		список	3.11,6.1
MAIN		главная	5.1
NAME		имя	6.2
NOCHECK		без проверки	7.3.3
NOCONVERSION	NOCONV	без преобразования	7.3.2

Продолжение

Ключевое слово	Сокращение	Перевод	Пункты
NOFIXEDOVER-FLOW	NOFOFL	без фиксированного переполнения	7.3.1
NOOVERFLOW	NOOFL	без переполнения	7.3.1
NOSIZE		без диапазона	7.3.1
NOSTRINGRAN-GE	NOSTRG	без границ строки	7.3.1
NOSUBSCRIPT-RANGE	NOSUBRG	без границ индекса	7.3.1
NOUNDERFLOW	NOUFL	без исчезновения	7.3.1
NOZERODIVIDE	NOZDIV	без деления на нуль	7.3.1
ON		при	7.3.2
OPEN		открыть	6.5.1
OPTIONS		дополнения	5.1
OUTPUT		вывод	6.5.1
OVERFLOW	OFL	переполнение	7.3.1,9.3.1
PAGE		страница	6.1,6.3.3
POINTER	PTR	указатель	7.1.3
POSITION	POS	позиция	4.5
PRINT		печатаемый	6.4.1
PROCEDURE	PROC	процедура	5.1, 5.2
PUT		выдать	3.11,6.1,6.3.4
READ		читать	6.5.1
REAL		действительный	1.7.1
RECORD		запись	6.0,6.5.1, 9.3.2
RECURSIVE		рекурсивный	5.4
REGIONAL		региональный	6.5.2
REPLAY		ответ	3.9
RETURN		возврат	5.1,5.2
RETURNS		возвращает	5.2,7.1.4
REWRITE		заменить	6.5.1
SEQUENTIAL	SEQL	последовательный	6.5.3
SIGNAL		имитация	7.3.2

Продолжение

Ключевое слово	Сокращение	Перевод	Пункты
SIZE		диапазон	3.1.1,7.3.1, 9.3.1
SKIP		прогон	3.11,6.1,6.3.3
STATIC		статический	7.1.3,7.1.4
STOP		стоп	3.10,5.1
STREAM		поток	6.0
STRING		строка	6.4.2
STRINGRANGE	STRG	границы строки	9.3.1
SUBSCRIPT-RANGE	SUBRG	границы индекса	7.3.1,9.3.1
SYSIN		системный ввод	6.4.1
SYSPRINT		системная печать	6.4.1
SYSTEM		системный	7.3.2
THEN		то	3.5
TITLE		название	6.5.1
TO		до	3.7,3.8
TRANSMIT		передача	7.3.1,9.3.2
UNALIGNED	UNAL	невыровненный	8.5.3
UNDEFINED-FILE		неопределенный файл	9.3.2
UNDERFLOW	UFL	исчезновение	7.3.1,9.3.1
UPDATE		исправлять	6.5.1
VARYING	VAR	изменяемый	8.2.1
WHILE		пока	3.8
WRITE		писать	6.5.1
ZERODIVIDE	ZDIV	деление на нуль	7.3.1,9.3.1

9.6. Символы ЕС ЭВМ

Символ	Код	Перфорация	Символ	Код	Перфорация	Символ	Код	Перфорация
пробел	40	нет	Ю	B8	12,11,0,8	Я	DD	12,11,9,8,5
[	4A	12,8,2	Б	BA	12,11,0,8,2	S	E2	0,2
.	4B	12,8,3	Ц	BB	12,11,0,8,3	T	E3	0,3
<	4C	12,8,4	Д	BC	12,11,0,8,4	U	E4	0,4
(	4D	12,8,5	Ф	BE	12,11,0,8,6	V	E5	0,5
+	4E	12,8,6	Г	BF	12,11,0,8,7	W	E6	0,6
!	4F	12,8,7	А	C1	12,1	X	E7	0,7
&	50	12	В	C2	12,2	Y	E8	0,8
]	5A	11,8,2	С	C3	12,3	Z	E9	0,9
¤	5B	11,8,3	D	C4	12,4	У	EB	11,0,9,8,3
*	5C	11,8,4	Е	C5	12,5	Ж	EC	11,0,9,8,4
)	5D	11,8,5	F	C6	12,6	Ь	EE	11,0,9,8,6
;	5E	11,8,6	G	C7	12,7	Ы	EF	11,0,9,8,7
¬	5F	11,8,7	Н	C8	12,8	ø	F0	0
—	60	11	I	C9	12,9	1	F1	1
/	61	0,1	И	CB	12,0,9,8,3	2	F2	2
|	6A	12,11	Й	CC	12,0,9,8,4	3	F3	3
,	6B	0,8,3	Л	CE	12,0,9,8,6	4	F4	4
%	6C	0,8,4	J	D1	11,1	5	F5	5

Продолжение

Символ	Код	Перфорация	Символ	Код	Перфорация	Символ	Код	Перфорация
—	6D	0,8,5	K	D2	11,2	6	F6	6
>	6E	0,8,6	L	D3	11,3	7	F7	7
?	6F	0,8,7	M	D4	11,4	8	F8	8
:	7A	8,2	N	D5	11,5	9	F9	9
#	7B	8,3	O	D6	11,6	З	FA	12,11,0,9,8,2
@	7C	8,4	P	D7	11,7	Ш	FB	12,11,0,9,8,3
'	7D	8,5	Q	D8	11,8	Э	FC	12,11,0,9,8,4
=	7E	8,6	R	D9	11,9	Щ	FD	12,11,0,9,8,5
"	7F	8,7	П	DC	12,11,9,8,4	Ч	FE	12,11,0,9,8,6

Шестнадцатеричная кодировка

16-я	10-я	2-ичная	16-я	10-я	2-ичная	16-я	10-я	2-ичная	16-я	10-я	2-ичная
0	0	0000	4	4	0100	8	8	1000	C	12	100
1	1	0001	5	5	0101	9	9	1001	D	13	1101
2	2	0010	6	6	0110	A	10	1010	E	14	1110
3	3	0011	7	7	0111	B	11	1011	F	15	1111

ПРИЛОЖЕНИЕ

ВЗАИМОСВЯЗЬ МОДУЛЕЙ PL/1 И ФОРТРАНА

В ряде случаев возникает необходимость объединить модули, написанные на разных языках программирования. Например, при разработке новых программ на PL/1 требуется использовать подпрограммы из пакета прикладных подпрограмм фортрана или старые модули, написанные ранее на фортране. Ниже описываются простые случаи объединения модулей, написанных на языках PL/1 и фортран, в ОС ЕС ЭВМ. С более сложными вопросами связи разноязыковых модулей можно познакомиться по имеющейся литературе [18, 20].

Информация между модулями в ОС ЕС ЭВМ может передаваться через аргументы (при обращении из одного модуля к другому по оператору CALL или с помощью указателя функции), через общие области памяти (EXTERNAL—в PL/1 и COMMON—в фортране), а также через наборы данных на внешних носителях. Однако в языке фортран имеется значительно меньше типов данных, чем в PL/1, а поэтому только ограниченное количество типов данных может быть передано из модуля PL/1 в модуль фортрана. Кроме того, в трансляторах PL/1 и фортрана ЕС по-разному организовано хранение в памяти ЭВМ соответствующих типов данных, а также по-разному реализованы соглашения о связях между модулями. Указанные различия приходится учитывать при составлении программ из разноязыковых модулей. В ряде случаев для этого используются модули-связки, написанные на языке ассемблера. В других случаях возможна непосредственная связь модулей.

1. Соответствие данных PL/1 и фортрана

При организации информационной связи между модулями, написанными на PL/1 и фортране, следует учитывать неполное соответствие типов данных этих языков, а в ряде случаев также различные способы представления и размещения данных в памяти.

Таблица 1

Соответствие типов данных PL/1 и фортрана ЕС

№	PL/1	Длина (в байтах)	Выравнивание	Фортран
1	FIXED BINARY (p, 0); $1 \leqslant p \leqslant 15$	2	полуслово	INTEGER *2
2	FIXED BINARY (p, 0); $16 \leqslant p \leqslant 31$	4	слово	INTEGER *4
3	FLOAT DECIMAL (p); $1 \leqslant p \leqslant 6$	4	слово	REAL *4
4	FLOAT BINARY (p); $1 \leqslant p \leqslant 21$	4	слово	REAL *4
5	FLOAT DECIMAL (p); $7 \leqslant p \leqslant 16$	8	двойное слово	REAL *8
6	FLOAT BINARY (p); $22 \leqslant p \leqslant 53$	8	двойное слово	REAL *8
7	COMPLEX FLOAT DEC (p); $1 \leqslant p \leqslant 6$	8	слово	COMPLEX *8
8	COMPLEX FLOAT BIN (p); $1 \leqslant p \leqslant 21$	8	слово	COMPLEX *8
9	COMPLEX FLOAT DEC (p); $7 \leqslant p \leqslant 16$	16	двойное слово	COMPLEX *16
10	COMPLEX FLOAT BIN (p); $22 \leqslant p \leqslant 53$	16	двойное слово	COMPLEX *16
11	BIT (8)	1	байт	LOGICAL *1
12	BIT (32)	4	байт	LOGICAL *4

Соответствие типов данных. Количество типов данных языка PL/1 значительно превышает их количество в фортране. Сведения о соответствии типов данных в PL/1 и фортране, приведенные ранее в п. 8.5.2, объединены в таблицу 1. Она может быть использована для проверки правильности описания данных взаимосвязанных модулей, написанных на PL/1 и фортране (о выравнивании данных см. п. 8.5.3).

Размещение массивов. Трансляторы PL/1 и фортрана по-разному размещают в памяти ЭВМ многомерные массивы: транслятор PL/1—«по строкам», а транслятор фортрана—«по столбцам». Таким образом, если массив передается из модуля фортрана в модуль PL/1, то при его описании в PL/1 следует изменить порядок следования индексов. Например, если в фортране описан массив A(5, 8, 10), то соответствующий ему массив в PL/1 должен быть описан как A(10, 8, 5).

Для того чтобы можно было работать с массивами, имеющими одинаковую индексацию в модулях PL/1 и фортрана, следует использовать для описания массива в PL/1 атрибут DEFINED и iSUB-переменные (см. п. 4.5).

Пример 1. Пусть описание массива в фортране имеет вид

```
REAL A(5, 8, 10)
```

Ему может соответствовать следующее описание в PL/1:

```
DECLARE A1(10, 8, 5) FLOAT,
        A(5, 8, 10) FLOAT DEFINED A1(3SUB, 2SUB,
        1SUB);
```

Массив A в этом случае будет иметь в программе на фортране и на PL/1 одну и ту же индексацию.

Представление данных. В модулях PL/1 данные, организованные в массивы и структуры, имеют информационные векторы, которых нет в фортране. *Информационный вектор* содержит истинный адрес массива или структуры и некоторые характеристики, описывающие эту величину. В процессе выполнения программы при обращении к переменной, описанной в модуле PL/1, на самом деле происходит обращение не к самой переменной, а к ее информационному вектору. Поэтому при передаче между модулями PL/1 и фортрана массивов и структур встает вопрос определения истинных адресов передаваемых данных.

Этот вопрос решается путем использования арифметической базированной переменной (см. п. 7.1.3), которая при размещении накладывается на начало передаваемой величины, а не на ее информационный вектор.

Пример 2. Пусть из модуля PL/1 требуется передать массив в модуль фортрана. Тогда модуль PL/1 может иметь следующий вид:

```
MODPL: PROCEDURE OPTIONS(MAIN);
       DECLARE PARM FIXED DECIMAL(1,0) BASED(P),
               AD(4, 3) FLOAT DECIMAL,
               ARRAY (3, 4) FLOAT DECIMAL
                            DEFINED AD(2SUB, 1SUB);
         P = ADDR(AD);
          . . . . . .
         S = ARRAY(I, J);
          . . . . . .
         CALL SF(PARM);
          . . . . . .
END MODPL;
```

Вызываемый модуль фортрана будет иметь вид

```
SUBROUTINE SF(ARRAY)
 REAL ARRAY(3, 4)
  . . . . . .
 ARRAY(2, 3) = D
  . . . . . .
END
```

Здесь в модуле MODPL при вызове подпрограммы SF в качестве аргумента используется арифметическая базированная переменная PARM. Переменная PARM будет расположена в памяти ЭВМ по тому же адресу, что и первый элемент массива AD. Это достигается наложением переменной PARM на массив AD с помощью указателя P, значение которого вычисляется встроенной функцией ADDR (см. п. 9.1.6). Массив AD, в свою очередь, совмещается (с перестановкой индексов) с массивом ARRAY, что позволяет в модулях PL/1 и фортрана использовать одинаковую индексацию массивов ARRAY.

Пример 3. Пусть требуется передать массив из модуля фортрана в модуль PL/1. Тогда модуль фортрана может иметь, например, следующий вид:

```
SUBROUTINE S
 DIMENSION ARRAY(15, 20, 45)
  . . . . . .
 CALL LINKPR(PLMOD, ARRAY)
  . . . . . .
END
```

Из этого модуля с помощью модуля-связки LINKPR происходит обращение к модулю PL/1 с именем PLMOD (о вызове модулей см. ниже), которому в качестве аргумента передается массив ARRAY.

Модуль PLMOD может иметь в этом случае следующий вид:

```
PLMOD: PROCEDURE(PARM);
       DECLARE PARM FIXED DECIMAL(1, 0),
               ARRAY(15, 20, 45) FLOAT DECIMAL,
               AD(45, 20, 15) FLOAT DECIMAL
                  DEFINED ARRAY(3SUB, 2SUB, 1SUB),
               ABAS(45, 20, 15) FLOAT DECIMAL BASED(P);
        P = ADDR(PARM);
        AD = ABAS;
        . . . . . .
        ABAS = AD;
END PLMOD;
```

В этом примере в модуле PLMOD параметром является арифметическая переменная PARM, адрес которой совпадает с адресом первого элемента массива ABAS. Необходимость введения дополнительного по сравнению с предыдущим случаем массива ABAS возникает из-за того, что существующий транслятор PL/1 не позволяет совмещаемой переменной быть базированной, то есть в данном случае нельзя базировать массив ARRAY, так как он совмещается с массивом AD.

В начале программы с помощью оператора присваивания значение массива ABAS присваивается массиву AD. В свою очередь, массив ARRAY совмещен с AD, и поэтому при обращении из модуля S к модулю PLMOD массив ARRAY фортрана совмещается с массивом ARRAY в PL/1.

2. Вызов подпрограмм

При использовании метода модульного программирования отдельные части программы на PL/1 или фортране оформляются в виде внешних процедур: процедур-подпрограмм или процедур функций Процедуры-подпрограммы вызываются с помощью операторов CALL, процедуры-функции — с помощью указателей функций.

При вызове подпрограмм фортрана из модуля PL/1 в операторе CALL указывается имя вызываемой подпрограммы. Во всех остальных случаях (при вызове процедур-подпрограмм и процедур-функций PL/1 из модуля фортрана, а также при вызове функций фортрана из модуля PL/1) обращение к вызываемому модулю осуществляется через модули-связки, написанные на языке ассемблера. При использовании модуля-связки в операторе CALL или в указателе функции указывается имя соответствующего модуля-связки, а имя вызываемой подпрограммы задается в качестве первого аргумента. Напря-

мер, для вызова процедуры-подпрограммы PL/1 из модуля фортрана может быть задано обращение вида

```
CALL LINKPR(PLMOD, A, B, C);
```

где LINKPR — имя модуля-связки, PLMOD — имя процедуры-подпрограммы PL/1, к которой требуется обратиться, A, B, C — аргументы, с которыми нужно обратиться к PLMOD.

Вызываемые модули должны быть описаны в вызывающем модуле (с атрибутами ENTRY в операторе описания PL/1 или в операторе EXTERNAL в фортране).

В задачу модуля-связки входит: установление среды языка, на котором написан вызываемый модуль, построение для него нового списка параметров, вызов требуемой подпрограммы или функции, восстановление старого списка параметров, возврат управления в вызывающий модуль.

Новый список параметров всегда строится для вызываемых процедур-функций. Это вызвано различиями в реализации вызова функций в языках PL/1 и фортран. В PL/1 для размещения результата выполнения функции строится дополнительная переменная, которая неявно присоединяется к списку параметров функции PL/1, а в фортране вычисленное значение функции помещается в общие регистры или регистры с плавающей точкой в зависимости от типа вызываемой функции. Для каждого типа возвращаемого значения должен быть написан свой модуль-связка. Ниже в качестве примера приводится текст модуля-связки PLREAL4, используемого для вызова процедуры-функции фортрана из модуля PL/1 в случае, когда возвращаемое значение имеет тип REAL*4 (FLOAT DECIMAL (6)).

Пример 4.

```
PLREAL4  CSECT
PLREAL4B DXD      A
         PRINT    NOGEN
         SAVE     (14,12)
         BALR     10,0
         USING    *,10
*ПОСТРОЕНИЕ ЦЕПОЧКИ DSA
         L        15,=V (IHESADA)
         LA       0,100
         BALR     14,15
         MVI      0(13),X'80'
*ПОСТРОЕНИЕ НОВОГО СПИСКА ПАРАМЕТРОВ
         LA       8,0(1)
M1       A        8,=F'4'
         L        4,0(8)
         ST       4,ADRPARM
```

```
         TM          ADRPARM,X'80'
         BZ          M1
         S           8,=F'4'
         OI          0(8),B'10000000'
*УСТАНОВЛЕНИЕ СРЕДЫ ФОРТРАНА
         L           15,=V(IBCOM#)
         BAL         14,64(15)
*ВЫЗОВ ФУНКЦИИ ФОРТРАНА
         L           15,0(1)
         L           15,0(15)
         LA          1,4(1)
         BALR        14,15
*ВОЗВРАТ РЕЗУЛЬТАТА В ВЫЗЫВАЮЩУЮ
*ПРОГРАММУ PL/1
         NI          0(8),B'01111111'
         L           8,4(8)
         STE         0,0(8)
*ВОЗВРАТ УПРАВЛЕНИЯ В ВЫЗЫВАЮЩУЮ ПРОГРАММУ
         L           15,=V(IHESAFA)
         BR          15
ADRPARM  DS          A
         END
```

Для других типов возвращаемых значений в этом модуле следует изменить команду

```
STE 0,0(8)
```

Например, для возвращаемого значения типа INTEGER*4 эта команда заменяется на

```
ST 0,0(8)
```

В качестве примера обращения из модуля PL/1 к подпрограмме фортрана рассмотрим использование в программе на PL/1 подпрограммы из пакета научных подпрограмм фортрана.

Пример 5. В программе на PL/1 требуется вычислить функцию Бесселя с обычной точностью $J_n(x)$.

В пакете научных подпрограмм фортрана имеется соответствующая подпрограмма BESJ, обращение к которой имеет вид

```
CALL BESJ(X, N, BJ, D, IER).
```

Здесь:

X — аргумент функции (тип — REAL * 4),
N — порядок функции (тип — INTEGER * 4),
BJ — вычисленное значение функции (тип — REAL * 4),
D — требуемая точность (тип — REAL * 4),
IER — индикатор ошибок (тип — INTEGER * 4).

Программа на PL/1, в которой вычисляется $J_n(x)$, может иметь, например, следующий вид:

```
PLBES: PROCEDURE OPTIONS(MAIN);
       DECLARE (N1, IER1) FIXED BINARY(31),
               (D1, BJ1, X1) FLOAT DECIMAL,
               BESJ ENTRY(FLOAT DEC,
                          FIXED BIN(31),
                          FLOAT DEC, FLOAT DEC,
                          FIXED BINARY(31));
        . . . . . .
        CALL BESJ(X1, N1, BJ1, D1, IER1);
        PUT LIST(BJ1);
        . . . . . .
END PLBES;
```

3. Обмен данными через общие области памяти

Для обмена данными между программами на PL/1 и фортране могут быть использованы поименованные общие области, описываемые в PL/1 структурой с атрибутом EXTERNAL, а в фортране — оператором COMMON.

Так как в языке PL/1 все данные, кроме скалярных арифметических переменных, имеют информационные векторы, то в начале общей области в модуле PL/1 располагается информационный вектор структуры, состоящий из информационных векторов массивов и адресов арифметических переменных, входящих в эту структуру. Поэтому для общей области в операторе COMMON модуля фортрана первым необходимо указывать фиктивный массив, размер которого равен количеству полных слов в информационном векторе структуры PL/1, а длина элементов этого массива должна быть равна 4 байтам. Переменные в общей области фортрана (и структуре PL/1) должны располагаться после фиктивного массива в порядке убывания их длин, в соответствии с требованиями транслятора фортрана.

Количество полных слов в информационном векторе PL/1 (размер соответствующего фиктивного массива в общей области фортрана) вычисляется по формуле

$$S = l + \sum_{i=1}^{k} M_i.$$

Здесь:

l — количество скалярных переменных, входящих в структуру (длина адреса каждой скалярной переменной — одно слово);

k — количество массивов, входящих в структуру PL/1;

M_j — длина информационного вектора j-го массива, которая определяется по формуле

$$M_j = 1 + 2 \cdot m_j;$$

m_j — размерность j-го массива.

Пример 6. Пусть общая область (структура) в PL/1 имеет вид

```
DECLARE 1 ST1 EXTERNAL,
          2 X COMPLEX FLOAT DECIMAL(6),
          2 A FLOAT DECIMAL(6),
          2 B(100, 20) FIXED BINARY(15),
          2 I FIXED BINARY(15);
```

Длина информационного вектора этой структуры равна

$$S = 3 + 1 + 2*2 = 8 \text{ слов.}$$

Соответствующее описание общей области в фортране им еет ви

```
COMMON /ST1/ DVAR, X, A, B, I
DIMENSION DVAR(8)
COMPLEX *8 X
REAL *4 A
INTEGER *2 B(100, 20), I
```

В описании общей области первым располагается фиктивный массив DVAR, длина которого 8 слов. За ним следуют переменные, расположенные в порядке убывания их длин.

4. Обмен данными через внешнюю память

Одним из способов передачи информации между модулями PL/1 и фортрана является обмен данными через внешнюю память. Данные, представленные в виде простых переменных, массивов и структур, могут передаваться между программами PL/1 и фортрана через наборы данных с последовательной организацией или через разделы библиотек.

При передаче данных из программы, написанной на языке фортран, в программу на языке PL/1 (и обратно) через внешние носители можно использовать как бесформатные (записеориентированные), так и форматные (потокоориентированные) операторы ввода-вывода, описанные в гл. 6. Преимущество следует отдать бесформатным операторам, поскольку данные в этом случае представляются на внешних носителях во внутренней форме, и никакого преобразования их во время выполнения операторов ввода-вывода не производится, что ускоряет передачу данных.

Напомним, что бесформатным операторам ввода-вывода фортрана WRITE(*a*) *c* и READ(*a*) *o*

соответствуют операторы PL/1

```
WRITE FILE(a) FROM(c);
```

и

```
READ FILE(a) INTO(c);
```

В общем случае *c* в фортране может быть списком переменных. Тогда каждому списку переменных фортрана соответствует структура в программе PL/1, состоящая из переменных, входящих в список. Следует учитывать, что при использовании бесформатных операторов ввода-вывода все передаваемые переменные в программе на фортране размещаются с границы байта. Поэтому соответствующие переменные в структуре PL/1 должны иметь атрибут UNALIGNED.

Так как в фортране ЕС при передаче данных посредством операторов бесформатного ввода-вывода могут использоваться только наборы данных с блоками переменной длины, допускающие сегментирование логических записей, то передавать данные между программами PL/1 и фортрана с помощью этих операторов можно только блоками, имеющими форматы записей VS и VBS.

Пример 7. Пусть через внешнюю память из программы PL/1 в программу на фортране с помощью бесформатных операторов передается структура R.

Программа PL/1 может иметь следующий вид:

```
EXP7: PROCEDURE OPTIONS(MAIN);
        DECLARE 1 R UNALIGNED,
                  2 U(100) FLOAT DECIMAL(16),
                  2 S(50) FIXED BINARY(15),
              ZAP FILE RECORD
                                ENVIRONMENT(VS(908));
        . . . . . .
        OPEN FILE(ZAP) OUTPUT;
        WRITE FILE(ZAP) FROM(R);
        CLOSE FILE(ZAP);
        . . . . . .
END EXP7;
```

Набор данных, в который записывается информация, описывается оператором DD с именем ZAP.

Программа на фортране, в которой данные считываются с внешнего носителя, имеет в этом случае такой вид:

```
REAL*8 U(100)
INTEGER*2 S(50)
. . . . . .
READ(9) U, S
. . . . . .
END
```

Набор данных, из которого считывается информация, должен быть описан оператором DD с именем FT09F001. Если модули фортрана и PL/1 обрабатываются в одном шаге задания, то соответствующий файл в модуле PL/1 опцией TITLE должен ссылаться на имя FT09F001.

Форматным операторам фортрана

WRITE(*a*, *b*) *c* и READ(*a*, *b*) *c*

соответствуют операторы PL/1

PUT FILE(*a*) EDIT(*c*) (R(*b*));

и

GET FILE(*a*) EDIT(*c*) (R(*b*));

Соответствующие форматы операторов ввода-вывода фортрана и PL/1 сведены в таблицу 2 (см. также п. 6.3).

С помощью форматных операторов ввода-вывода можно передавать данные между программами PL/1 и фортрана блоками, имеющими форматы записей F, FB, V, VB, U.

Таблица 2

Соответствие форматов операторов ввода-вывода в PL/1 и фортране

Фортран	PL/1
I*w*	F(*w*)
F*w*.*d*	F(*w*,*d*)
E*w*.*d*	E(*w*,*d*)
D*w*.*d*	E(*w*,*d*)
A*w*	A(*w*)

Здесь *w* — количество позиций, в которых размещается значение; *d* — количество позиций, в которых размещается дробная часть числа.

При обмене данными между программами на языках PL/1 и фортран с помощью форматных операторов ввода-вывода, следует учитывать следующие различия во внешнем представлении чисел с плавающей точкой в этих языках:

— показатель степени числа с плавающей точкой двойной точности указывается в языке PL/1 литерой E, а в фортране — литерой D;

— если показатель степени положителен, то в PL/1 он имеет знак «+», а в фортране знак «+» заменяется пробелом.

В силу указанных причин с помощью форматных операторов ввода-вывода в PL/1 можно принимать из фортрана только числа с отрицательным порядком и имеющие точность $p \leqslant 6$.

Пример 8. Пусть требуется передать из модуля фортрана в модуль PL/1 данные через внешние носители, используя форматные операторы ввода-вывода. Тогда модуль фортрана может иметь, например, следующий вид:

```
      INTEGER*2 K(40)
      REAL*4 B(100)
      . . . . . .
       WRITE(9, 20) K, B
20      FORMAT(40I6, 100E13.5)
        . . . . . .
      END
```

Модуль PL/1, в котором используются данные, записанные на внешний носитель, будет в этом случае иметь такой вид:

```
ACS: PROCEDURE OPTIONS(MAIN);
     DECLARE B(100) FLOAT DECIMAL(6),
             K(40) FIXED BINARY(15),
        PLFILE FILE RECORD INPUT;
     . . . . . .
     GET FILE(PLFILE) EDIT(K, B) (R(LF));
   LF: FORMAT(40 F(6), 100 E(13, 5));
. . . . . .
END ACS;
```

В этом примере данные будут передаваться правильно только в том случае, если элементы массива B будут иметь отрицательный порядок.

5. Замечания по составлению заданий

При составлении заданий на трансляцию, редактирование и выполнение программ, состоящих из модулей, написанных на разных языках, следует учитывать следующее:

1. Для пункта редактирования следует предусмотреть подключение библиотек трансляторов PL/1 (SYS1.PL1LIB) и фортрана (SYS1.FORTLIB), то есть в задании должны присутствовать операторы

```
//LKED.SYSLIB DD DSNAME=SYS1.PL1LIB,DISP=SHR
//            DD DSNAME=SYS1.FORTLIB,DISP=SHR
```

Если из программы происходит обращение к пакету научных подпрограмм фортрана PNPLG, то к этим предположениям следует добавить оператор DD, описывающий эту библиотеку:

// DD DSNAME=PNPLG,DISP=SHR

2. Если для выполнения программы используется каталогизированная процедура PL1LFCLG, то шаг выполнения следует дополнить оператором DD, обеспечивающим вывод на печать данных в модулях фортрана:

/GO.FT06F001 DD SYSOUT=A
/

При использовании каталогизированной процедуры FORTGCLG шаг выполнения следует дополнить оператором

//GO.SYSPRINT DD SYSOUT=A

обеспечивающим вывод на печать данных в модулях PL/1.

ОТВЕТЫ К УПРАЖНЕНИЯМ

1.1. а) TEMA31; б) ▢1; в) ATAN(Z); г) 1/TAN(ALPHA);
д) 3.5E0+15.1E0I ; е) 40E−5I ; ж) '␣ПЕЧАТЬ␣'''

1.2. б), в), д).

1.3. б), в), е), ж), к), л).

2.1. а) 3; б) 8; в) 10.

2.2. а) (R(I)+R(I+1))/C1/C2+(A**K−X/(C@+0.5))
/(0.5+X*X/(R(I−1)−R(I)))
б) R1/R2*(SQRT(BETA)−EXP(3E−5**X**0.3)
+R2*COS((PI*Y)**(1/3)))**X**(1/7)/2/(1−R#)

2.3. а) X*Y¬=0 & ¬B; б) X < 1 ! ¬(Y=Z) & Z¬=X

2.4. а) **ложь**; б) **ложь**; в) **истина**; г) **истина**; д) **ложь**;
е) **ложь**; ж) **истина**; з) **ложь**; и) **истина**; к) **ложь**; л) **ложь**;

3.1. а) 7.89000E3; б) 0123; в) 5.00000E−1; г) 0000.

3.2.
```
DECLARE G(4) LABEL INITIAL(L140, L170, L190, L200);
GO TO G(N);
```

3.3.
```
DECLARE F(4) LABEL INITIAL(L#1, L#2, L#3, L#4);
GO TO F(K);
L#1: Y=X+0.5;            GO TO CZ;
L#2: Y=X*X*X−1E−4;       GO TO CZ;
L#3: Y=B*X*X;            GO TO CZ;
L#4: Y=X/B;
CZ: Z=0.3*Y*Y−B;
```

3.4.
```
IF X<Y THEN T=Y; ELSE T=X; V=T*T;
```

3.5.
```
IF X>Y THEN M=X; ELSE M=Y;
IF Z>M THEN M=Z;
```
или
```
IF X>Y & X>Z THEN M=X;
ELSE IF Y>Z THEN M=Y; ELSE M=Z;
```

3.6.
```
US=1;
DO N=1 TO 100;
 US=US*N;
END;
```

3.7.
```
SUMVEC=0;
DO I=1 TO N;
 SUMVEC=SUMVEC+A(I);
END;
```

```
3.8.      SUM=0;
      SM: DO I=1 TO N;
            DO J=1 TO M;
              SUM=SUM+A(I,J);
            END;
      END SM;
3.9.  UMN, UMP=1;
      DO K=1 TO N;
        IF C(K) < 0 THEN UMN=UMN*C(K);
        IF C(K) > 0 THEN UMP=UMP*C(K);
      END;
3.10. GET LIST(X, Y, Z);
      PUT LIST(X, Y, Z, 'МАКСИМУМ ␣=', M);
3.11. GET LIST(N); ...; GET LIST(A);
      PUT LIST('N␣=', N); PUT LIST('A␣=', A) SKIP;
      PUT LIST('СУММА␣ВЕКТОРА␣=', SUMVEC) SKIP;
3.12. GET LIST(N, M); ...; GET LIST(A);
      PUT LIST(N, M); PUT LIST(A) SKIP;
      PUT LIST('СУММА␣МАТРИЦЫ␣ДЛЯ',
                                    'N␣=', N, 'M␣=', M) SKIP;
      PUT LIST(SUM) SKIP;
3.13. DO M=1 TO N WHILE(Q(M) >=0); PUT LIST(Q(M)); END;
3.14. GET LIST(NUMBER);
      DO WHILE(NUMBER¬=0);
        PUT LIST(NUMBER);
        GET LIST(NUMBER);
      END;
3.15. YN#1=0.5+0.5*X;
      YN=0.5*X; /* ДЛЯ ВХОДА В ЦИКЛ */
      DO WHILE(ABS(YN#1-YN) > 1E-8);
        YN=YN#1; /* ПЕРЕХОД К СЛЕД. ИТЕРАЦИИ */
        YN#1=0.5(YN+X/YN);
      END;
4.1.  A(7) FIXED, B(7) FLOAT,
                          C(5) COMPLEX FLOAT(8) INITIAL((5)0),
      D(5) FLOAT(8) INITIAL((5)0), E FLOAT(8).
4.2.  а) INITIAL((21) (80)'1'B, (399) (80)'0'B, (21) (80)'1'B);
      б) INITIAL((21) ((40)'10'B, (19) (40)'01'B, (40)'10'B));
      в) INITIAL((20) (1)'0'B,(21) ('1'B, (19) (1)'0'B), '0'B);
4.3.  DECLARE Q(1:30,1:30) FLOAT,
                (I, J, IM, JM, MAX_Q) FIXED;
       GET LIST(Q); PUT LIST('Q',Q) SKIP;
       MAX_Q=Q(1,1); IM, JM=1;
```

```
    DO I=1 TO 30;
      DO J=1 TO 30;
        IF A(I,J) > MAX_Q
          THEN DO; MAX_Q=A(I,J); IM=I; JM=J; END;
      END; /* ПО J*/
    END; /* ПО I */
    PUT LIST('IM, ␣JM␣=', IM, JM,
                                  'MAX_Q␣=', MAX_Q) SKIP;
```

4.4.
```
DECLARE (B(1:40), S) CHAR(25),
        (I,J) FIXED;
GET LIST(B);
PUT LIST('ИСХОДНЫЙ␣МАССИВ'); PUT LIST(B) SKIP;
DO J=1 TO 39;
  DO I=1 TO 40-J;
    IF B(I) > B(I+1) THEN
      DO; S=B(I+1); B(I+1)=B(I); B(I)=S; END;
    /* END IF */
  END/* I */;
END/* J */;
PUT LIST('УПОРЯДОЧЕННЫЙ␣МАССИВ') SKIP(2);
PUT LIST(B) SKIP;
```

5.1.
```
MAXQ: PROCEDURE OPTIONS(MAIN);
```
Далее см. ответ к 4.3.
```
END MAXQ;
```

5.2.
```
TRANSP: PROCEDURE OPTIONS (MAIN);
  DECLARE T (1:20,  1:20) FLOAT (10),
    S (1:20, 1:20) FLOAT (10) DEFINED T (2SUB, 1SUB);
  GET LIST (T);
  PUT LIST ('ИСХОДНАЯ␣МАТРИЦА␣=␣', T) SKIP;
  PUT LIST ('ТРАНСПОНИРОВАННАЯ', S) SKIP (3);
END TRANSP;
```

5.3.
```
UMPN: PROCEDURE (C, UMP, UMN, N);
        DECLARE (C(*), UMP, UMN) FLOAT,
                (N, K) FIXED;
```
Далее см. ответ к 3.9.
```
END UMPN;
```
Описание входа:
```
DECLARE SUMPN ENTRY ((*) FLOAT,
                                  FLOAT, FLOAT, FIXED);
```

5.4.
```
SUMMAT: PROCEDURE (A, N, M) RETURNS (FLOAT (12));
          DECLARE (A (*, *), SUM) FLOAT (12),
                  (N, M, I, J) FIXED;
```

Далее см. ответ к 3.8.

```
            RETURN (SUM);
END SUMMAT;
```

5.5.
```
COS: PROCEDURE (X, EPS) RETURNS (FLOAT);
     DECLARE (X, EPS, U, S) FLOAT, I FIXED;
      U=1; S=1;
      DO I=1 BY 2 WHILE (ABS(U) >=EPS);
        U=-U*X*X/I/(I+1);
        S=S+U;
     END;
     RETURN (S);
END COS;
```

описание входа:

```
DECLARE COS ENTRY (FLOAT, FLOAT)  RETURNS (FLOAT);
```

6.1.
```
ON ENDFILE (SYSIN) NEXT='НЕТ';
PUT LIST ((47)'␣', 'ВИД␣ПЕРФОКАРТ') SKIP;
NEXT='ДА';
GET LIST (S, T) SKIP;
DO K=1 BY 1 WHILE (NEXT='ДА');
   PUT LIST ((23) '␣', K, S, T) SKIP;
   GET LIST (S, T) SKIP (2);
END;
```

6.2.
```
GET DATA (A);
DO J=1 TO 20;
   PUT DATA ((A(I,J) DO I=1 TO 4)) SKIP;
END;
```

6.3.
```
PUT EDIT ((69) '-') (COLUMN (11), A) PAGE;
PUT SKIP EDIT ('I', 'X', 'I', 'SIN␣X', 'I',
                                  'COS␣X', 'I','␣TG␣X', 'I')
                  (COL (10), A, COL (15), A, COL (20), 7 (A, X (7)));
PUT SKIP EDIT ('I', 'I', 'I', 'I', 'I', (69) '-')
   (COL(10), A, COL(20), A, COL(40), A, COL(60), A,
                                  COL(80), A, SKIP, COL (11), A);
PUT EDIT (('I', X, 'I', SIND(X), 'I', COSD(X), 'I', TAND(X), 'I'
                                  DO X=0 BY 1 TO 45))
   (COL (10), A, X(4), F(2), COL(20), A, F(3(X(4), F(11,8), X(4), A));
PUT EDIT ((69) '-') (COLUMN (11), A) SKIP;
```

6.4.
```
GRAFIC: PROCEDURE (F, M, A, B);
     DECLARE F ENTRY (FLOAT) RETURNS (FIXED),
             (A, B) FLOAT,
             (M, K, FUN) FIXED,
             (G (0:M, 0:100) CHAR (1)
                  INITIAL((M)('I', (100)(1)'␣'), (101)(1) '-');
```

```
        G='␣';
        DO K=0 TO 100 BY 1;
           FUN=F(A+K*(B-A)/100);
           G(M-FUN, K)='*';
        END;
        PUT EDIT(G) (SKIP, (101) A(1)) PAGE;
END GRAFIC;
```

Описание входа:

```
DECLARE GRAFIC ENTRY (ENTRY(FLOAT)
                      RETURNS(FIXED), FIXED, FLOAT, FLOAT);
```

6.5.
```
DECLARE A CHAR(7), B CHAR (13),
        C CHAR (17), CTP CHAR (40);
PUT STRING(CTP) EDIT (A, B, C) (A(5), X(3), A(11), X(3), A(15));
```

6.6.
```
DECLARE FS FILE INPUT RECORD ENVIRONMENT (F(24)),
        FR FILE OUTPUT DIRECT
                    ENVIRONMENT (REGIONAL (1) F(21)),
        RECS CHAR (24), RECR CHAR (21),
        NUM CHAR (3), CONT BIT (1);
 ON ENDFILE (FS) CONT='0'B;
  CONT='1'B;
  DO WHILE (CONT);
       READ FILE (FS) INTO (RECS);
       IF CONT THEN DO;
       GET STRING (RECS) EDIT (NUM, RECR) (A(3), A(21));
       WRITE FILE (FR) FROM (RECR) KEYFROM (NUM);
                    END;
  END;
```

7.1.
```
RAZ: PROCEDURE RETURNS (FIXED (6));
     DECLARE K FIXED (6) STATIC INITIAL (0);
       K=K+1;
       RETURN (K);
END RAZ;
```

7.2.
```
MAXIJ: PROCEDURE OPTIONS (MAIN);
       DECLARE (II, JJ) FIXED;
         GET LIST (II, JJ);
         PUT LIST ('II=', II, 'JJ=', JJ);
         BEGIN;
```

Далее см. ответ к 4.3, в котором числа, равные 30, должны быть заменены в одном случае на II, а в другом — на JJ.

```
         END;
END MAXIJ;
```

7.3.
```
REGULAR: BEGIN;
    DECLARE (B (NL : NR), S) CHAR (L) CONTROLLED,
            (I, J) FIXED;
     GET DATA (NL, NR, L);
     ALLOCATE B, S;
```
Далее см. ответ к 4.4 без описаний и с заменой в DO-операторах 1 на NL, 40—на NR, 39—на NR—1.
```
     FREE B, S;
END REGULAR;
```

7.4.
```
DECLARE 1 EXAMINE (10, 6),
          2 (OTL, HOR, UD, NEUD, NO) FIXED (3),
        1 EX_GROUP (10) LIKE EXAMINE,
        1 EX_MARK (6) LIKE EXAMINE,
        1 EX_COURSE LIKE EXAMINE,
        (G, M, C) FIXED;
        EX_GROUP=0; EX_MARK=0; EX_COURSE=0;
        /*СПИСОК ЛЕВОЙ ЧАСТИ НЕВОЗМОЖЕН,
      ТАК КАК СТРУКТУРЫ РАЗНОЙ ОРГАНИЗАЦИИ*/
        DO G=1 TO 10; DO M=1 TO 6;
          EX_GROUP(G)=EX_GROUP(G)+EXAMINE(G,M);
          EX_MARK (M)=EX_MARK (M)+EXAMINE(G, M);
        END; END;
        DO M=1 TO 6;
          EX_COURSE=EX_COURSE+EX_MARK (M);
        END; /* СЕЧЕНИЯ МАССИВОВ
                        СТРУКТУР НЕ ДОПУСКАЮТСЯ! */
```

7.5.
```
(CHECK (X, Y)):
PROG: PROCEDURE OPTIONS (MAIN);
      . . . . . .
      ON CHECK (X) PUT DATA (X, Y, M);
      ON CHECK (Y) PUT DATA;
      . . . . . .
      (NOCHECK (Y)):
      B: BEGIN;
          ON CHECK (X) SYSTEM;
          ON OVERFLOW
              BEGIN; ON ERROR SYSTEM;
                     PUT DATA;
              END;
          . . . . . .
      END B;
      . . . . . .
END PROG;
```

```
8.1. MULTMAT: PROCEDURE (A, B, C);
                DECLARE (A, B, C) (*, *) FLOAT,
                        (I, J) FIXED BINARY;
                  DO I = 1 TO HBOUND (A, 1);
                    DO J = 1 TO HBOUND (B, 2);
                       C (I, J) = SUM (A(I, *) * B(*, J));
                    END;
                  END;
    END MULTMAT;
8.2. INTSTR: PROCEDURE (STR);
              DECLARE STR CHAR (*),
                      POS FIXED BINARY;
               POS = INDEX (STR, '.');
               IF POS ¬= 0 THEN SUBSTR (STR, POS) = '␣';
    END INTSTR;
8.3. EXCHAN: PROCEDURE (CTP, KOL);
               DECLARE CTP CHAR (*),
                       (KOL, L) FIXED BINARY,
                       S CHAR (KOL);
               L = LENGTH (CTP);
               S = SUBSTR (CTP, 1, KOL);
               SUBSTR (CTP, 1, KOL) =
                                SUBSTR (CTP, L - KOL + 1, KOL);
               SUBSTR (CTP, L - KOL + 1, KOL) = S;
    END EXCHAN;
8.4. EQLINE: PROCEDURE (F, G) RETURNS (BIT(1));
              DECLARE (F, G) (*, *) FIXED,
                      E (LBOUND (F, 1): HBOUND (F, 1)) BIT (1),
                      I FIXED BINARY;
               DO I = LBOUND (F, 1) TO HBOUND (F, 1);
                  E(I) = ANY (F(I, *) ¬= G (I, *));
               END;
               RETURN (ALL (E));
    END EQLINE;
```

ЛИТЕРАТУРА

1. Дал У., Дейкстра Э., Хоор К. Структурное программирование.— М.: Мир, 1975.
2. Вирт Н. Систематическое программирование.— М.: Мир, 1977.
3. Йодан Э. Структурное проектирование и конструирование программ.— М.: Мир, 1979.
4. Хьюз Дж., Мичтом Дж. Структурный подход к программированию.— М.: Мир, 1980.
5. Гребенников Л. К., Лебедев В. Н. Решение задач на ПЛ/1 в ОС ЕС.— М.: Финансы и статистика, 1981.
6. Безбородов Ю. М. Сравнительный курс языка PL/1 (на основе Алгола-60).— М.: Наука, 1980.
7. Программирование на ПЛ/1 ОС ЕС/Аугустон М. И., Балодис Р. П. и др.— М.: Статистика, 1979.
8. Бухтияров А. М., Фролов Г. Д., Олюнин В. Ю. Сборник задач по программированию на языке ПЛ/1.— М.: Наука, 1983.
9. Грунд Ф. Программирование на языке фортран IV.— М.: Мир, 1976.
10. Салтыков А. И., Макаренко Г. И. Программирование на языке фортран.— М.: Наука, 1984.
11. Брич З. С., Капилевич Д. Б., Котик С. О., Цагельский В. И. Фортран ЕС ЭВМ.— М.: Статистика, 1978.
12. Бухтияров А. М., Маликова Ю. П., Фролов Г. Д. Практикум по программированию на фортране.— М.: Наука, 1983.
13. Язык программирования фортран. ГОСТ 23056-78.— М.: Издательство стандартов, 1978.
14. Хьюз Ч., Пфлигер Ч., Роуз Л. Методы программирования: курс на основе фортрана.— М.: Мир, 1981.
15. Катцан Г. Язык фортран 77.— М.: Мир, 1982.
16. Средства отладки программ в ОС ЕС ЭВМ./Ерофеев В. И., Маркушев Ю. П., Першиков В. И., Соколов А. П.— М.: Статистика, 1979.
17. Безбородов Ю. М. Индивидуальная отладка программ.—М.: Наука, 1982.
18. Лебедев В. Н., Соколов А. П. Введение в систему программирования ОС ЕС.— М.: Статистика, 1978.
19. Эшли Р., Фернандес Д. Язык управления заданиями.— М.: Мир, 1981.
20. Лаврищева Е. М., Грищенко В. Н. Связь разноязыковых модулей в ОС ЕС.— М.: Финансы и статистика, 1982.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ